Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Adquisición y transmisión de vídeo 3D sin comprimir sobre redes IP

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Oscar Fernández Ardanuy

DIRECTOR: David Rincón Rivera

FECHA: 20 de Mayo de 2011

Título: Adquisición y transmisión de vídeo 3D sin comprimir sobre redes IP

Autor: Oscar Fernández Ardanuy

Director: David Rincón Rivera

Fecha: 20 de Mayo de 2011

Resumen

Este documento describe el desarrollo y puesta en marcha de un escenario de vídeo 3D en tiempo real, que incluye desde la adquisición de las imágenes con cámaras 3D hasta la visualización en un reproductor, pasando por la transmisión sobre redes IP.

Se han utilizado dos cámaras 3D con dos objetivos, puertos y tecnologías diferentes. La más modesta utiliza el puerto USB y la otra, más profesional, el IEEE 1394. Aunque cada una funcione de un modo distinto, en los dos casos las transmisiones de contenido se han implementado siguiendo el RFC 4175 para vídeo sin comprimir sobre RTP. Las imágenes se transmiten en YUV 4:2:2 en modo "Side-by-Side", que corresponde a juntar las dos imágenes en una sola imagen 2D.

También se ha desarrollado un repetidor RTP (en modo unicast) para distribuir el flujo a todos los clientes que lo demanden, así como la señalización que el repetidor y el reproductor intercambian para gestionar la comunicación.

El reproductor desarrollado muestra las imágenes sin realizar ningún tipo de efecto 3D, ya que ese es el objetivo de otro TFC desarrollado en paralelo. Permite seleccionar redimensionarlas si se reciben siguiendo el formato de ahorro de ancho de banda para "Side-by-Side". Este formato se basa en reducir a la mitad las columnas de cada imagen.

El principal hándicap del proyecto ha sido conseguir el rendimiento necesario para funcionar en tiempo real. Se han ido encontrando soluciones a lo largo del desarrollo que se comentan junto a sus resultados.

El uso del estándar RFC 4175 posibilita la fácil adaptación de los módulos del escenario a otros escenarios, o la de otros módulos a este escenario de transmisión 3D extremo a extremo.

Title: Acquisition and transmission of uncompressed 3D video over IP networks

Author: Oscar Fernández Ardanuy

Director: David Rincón Rivera

Date: May, 20th 2011

Overview

This document describes the development and implementation of a 3D video scenario on real time, including the acquisition of images with a 3D camera its transmission over an IP network, and its display on a player.

We have used two 3D cameras with different optics, ports and technologies. The modest one uses an USB port while the other, more professional, uses IEEE 1394. Although each camera works differently, both have been adapted to work with the RFC 4175 standard for uncompressed video delivery over IP networks. Images are coded in the YUV 4:2:2 scheme in “Side-by-Side” mode, merging both sources in a single 2D image.

We have developed also a repeater, in order to distribute the flow among all clients requesting a transmission. The repeater and the player exchange signaling messages for controlling the session.

The player displays the images without any kind of 3D effect – this is the target of another project being developed in parallel. It allows selecting the images to be resized if they are received following the bandwidth-saver format for “Side-by-Side”. This format halves the columns of each image.

The main handicap of this project has been to achieve the efficiency and performance level needed in order to operate in real time. This document describes the solutions and decisions chosen throughout the development, along with the results obtained.

The use of the RFC 4175 standard allows an easy adaptation of the modules and its integration with other projects.

ÍNDICE

INTRODUCCIÓN

El mundo multimedia se caracteriza por estar en constante evolución. La intención es conseguir una experiencia lo más realista posible para nuestros sentidos. Hasta la fecha se puede afirmar que en el apartado visual se ha llegado a un nivel de detalle excepcional, y la industria, poco a poco, va buscando otras maneras de evolucionar. El Blu-ray lleva unos años ofreciendo vídeo en Full HD (1080p), también se van haciendo pruebas en televisión en HDTV (1080i y 720p) y, mientras, se desarrollan sistemas Ultra HD (4K o 4320p) [R1] para substituir a los actuales tanto en disco como por televisión (UHDTV) [R2]. Del mismo modo que en el apartado de sonido se fue mejorando la calidad, y después se fue buscando sistemas envolventes, en el campo de la TV se está apostando por el contenido en 3D.

El vídeo en 3D se ha intentado integrar en muchas ocasiones como una evolución más, pero comercialmente nunca ha llegado a cuajar. El problema de necesitar elementos especiales (gafas, monitores) y no contar con la suficiente tecnología han sido dos losas que lo han condenado al fracaso en los intentos anteriores. Actualmente, y en buena parte gracias a la digitalización, ya completada, de la cadena de distribución de TV, la tecnología ha avanzado lo suficiente como para empezar a ofrecer vídeo en 3D de calidad y de manera asequible.

Si en el audio se consigue la sensación estéreo (2 canales) o envolvente (5 o más canales) con la instalación de altavoces en puntos estratégicos alrededor del oyente, para el vídeo en 3D se necesita que cada ojo del espectador reciba imágenes diferentes. Estas imágenes tienen que ser complementarias y correspondientes a un mismo instante, de manera que el cerebro las pueda integrar y formar una imagen con volumen.

Figura 0.1 Paralelismo entre audio y vídeo y el número de canales. Mono, estéreo, envolvente.

En la figura 0.1 queda reflejado el paralelismo entre el audio y el vídeo y su evolución. Se puede observar que el vídeo apenas ha avanzado en la utilización de múltiples canales para ofrecer sensaciones envolventes. Mientras, el audio ha aprovechado su mayor facilidad de desarrollo para progresar en este punto. Actualmente el vídeo en 3D empieza a entrar en las casas de los usuarios con sistemas que necesitan 2 canales para funcionar. Con 2 canales el vídeo puede ofrecer sensación de profundidad, pero con más podría llegar a conseguir imágenes realmente 3D que se pudiesen ver desde todos los ángulos, es decir, vídeo envolvente. Ya existen algunos desarrollos en este sentido [R3].

Este proyecto se ha centrado en un escenario de recepción, transmisión y reproducción de vídeo 3D. Todo esto en tiempo real, sin comprimir el flujo, sobre redes IP y haciendo uso de protocolos estándar para facilitar la integración en otros entornos, o de otros elementos en nuestro escenario.

El primer paso ha sido conseguir el flujo de vídeo 3D. Para ello se necesita una cámara que ofrezca un mínimo de dos puntos de vista, o lo que es lo mismo, con dos ópticas. En el proyecto se han utilizado un par de cámaras orientadas a dos segmentos de mercado distintos: una económica destinada a mercado de consumo y otra más profesional.

Para controlar cada cámara se ha utilizado un software distinto. Se han aprovechado aplicaciones que ya controlaban las cámaras y se ha añadido un módulo de transmisión, único y válido para las dos cámaras, siguiendo el RFC 4175 [N4] para vídeo sin comprimir. De este modo se consigue estandarizar la transmisión. En la figura 0.2 se muestra una pequeña representación de lo que se podría llegar a hacer siguiendo este modelo.

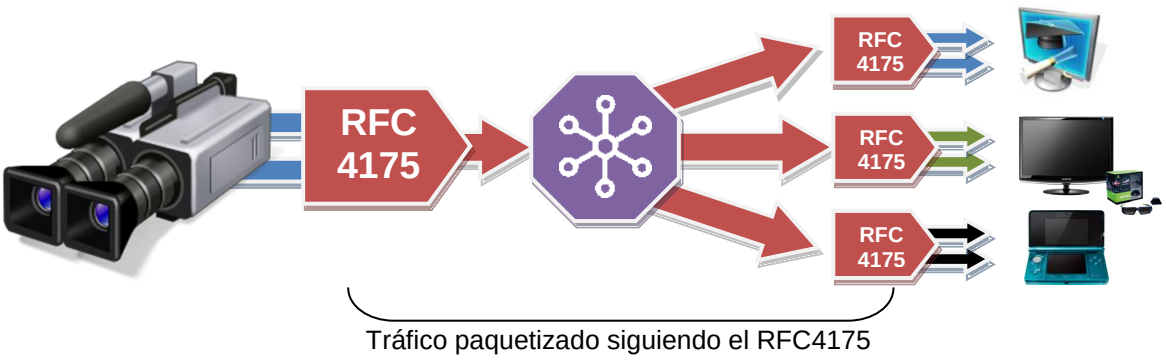


Figura 0.2 Escenario de transmisión siguiendo el estándar RFC 4175, con reproductores de diferentes tipos y capacidades de presentación

El RFC 4175 indica las especificaciones a la hora de transmitir vídeo sin comprimir sobre redes IP mediante el protocolo de tiempo real RTP. Pero no está preparado para vídeo 3D, por lo que algunos campos como el que indica el formato que contiene el vídeo se han tenido que extender siguiendo un criterio propio. No nos consta, siquiera, que haya ningún proyecto de modificación del RFC 4175 o algún draft de estándar nuevo para esta situación.

Al no estar definida todavía la política de transmisiones de vídeo 3D sin comprimir, se ha optado por tratar cada pareja de imágenes como si de una sola se tratase. Se transmiten en formato "Side-by-Side", que no es más que tomar una captura de cada objetivo y unirlas en un mismo plano eliminando la mitad de las columnas de cada una. Así se consigue de forma sencilla asegurar su sincronía pero se pierde bastante calidad. También se aceptan varias resoluciones y un formato "Side-by-Side extendido" donde se conservan todas las columnas. En la figura 0.3 se puede ver un ejemplo de cada formato.

Figura 0.3 Ejemplo de dos parejas de imágenes, la 1ª en "Side-by-Side" y la 2ª en "Side-by-Side extendido"

También se ha implementado un repetidor para distribuir copias de flujos, si fuera necesario. Se ha adaptado el repetidor de paquetes "RTP Unicast Mirror" [N6] para funcionar según el criterio deseado, haciéndolo compatible con la señalización ideada para comunicar el transmisor con el reproductor.

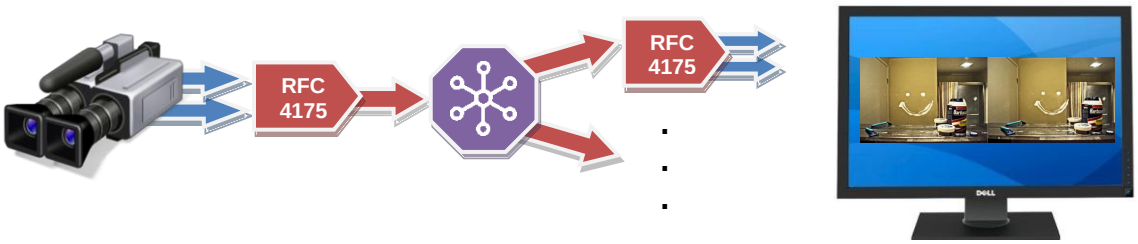
El reproductor implementado también se ha diseñado siguiendo el RFC 4175 y la señalización mencionada. El reproductor no muestra el vídeo con efecto 3D, sino que presenta las imágenes en pantalla tal cual le llegan, aunque permite reescalar las que llegan en "Side-by-Side". En otro TFC se está trabajando para suplir esta carencia y poder representar las imágenes con efecto 3D.

El hecho de ser un escenario, cuya principal característica es la transmisión de vídeo sin comprimir en tiempo real, genera un claro inconveniente: el ancho de banda necesario, que como veremos más adelante, será del orden de centenares de Mbit/s. Esto supondrá, por un lado, problemas a la hora de tener que tratar tal volumen de información, tanto en su procesado en tiempo real como en su transmisión, que deberá hacerse sobre una red de al menos 1 Gbit/s (Gigabit Ethernet).

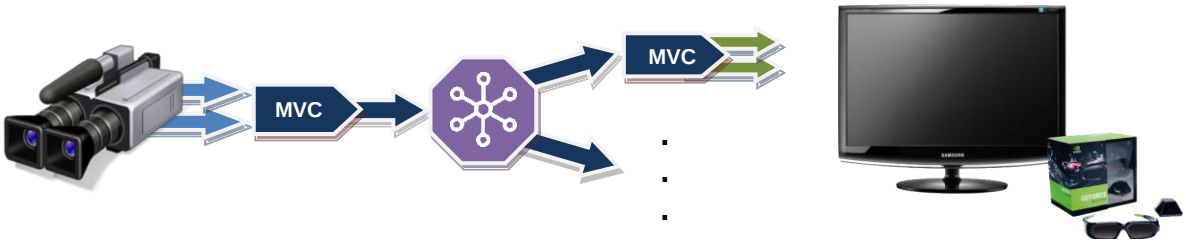
Por otro lado, el lado bueno de no comprimir es que no hay pérdidas de calidad por el codificador, y que nos ahorramos el coste computacional del compresor, que puede llegar a ser muy alto. Estos retos se tratarán a lo largo del documento, así como las posibles soluciones que fueron surgiendo durante el desarrollo del proyecto. El documento también incluye todas las políticas que se han seguido para realizar cada módulo y posibles mejoras y/o ampliaciones futuras.

Este tipo de escenario es innovador, tanto por el apartado de recepción con varias cámaras estereoscópicas, como el hecho de transmitir el vídeo sin comprimir y utilizando un repetidor para transmitir en modo unicast. Las únicas referencias que se tienen de sistemas similares se encuentran en sistemas

dedicados a investigación, como es el caso de UltraGrid [anexo D]. Esto ha hecho que la labor de desarrollar y, sobretodo, lograr un buen rendimiento en el escenario requiriese de muchas pruebas, e ir buscando cómo mejorar el rendimiento de cada elemento.



Escenario actual, siguiendo el RFC4175 para vídeo sin comprimir y sin reproductor 3D



Escenario futuro, con el vídeo comprimido y reproductor 3D con el kit de nVIDIA

Figura 0.4 Comparativa entre el escenario actual y futuras ampliaciones por otros proyectos

Este proyecto forma parte de un escenario más amplio en el que otros estudiantes han desarrollado o están desarrollando diferentes módulos. En el pasado se han realizado TFCs relacionados con la captación y visualización de vídeo 3D [R4], así como con la compresión en tiempo no real de vídeo 3D mediante el codificador H.264/MVC [R5]. En estos momentos se está trabajando en las líneas de compresión MVC en tiempo real [R6], en la visualización en 3D con gafas obturadoras y monitor de 120 Hz [R7], y la transmisión de 3D sobre redes peer-to-peer (P2P) [R8]. En la figura 0.4 hay una muestra de la evolución que puede tener el escenario base del presente proyecto, cuando se finalicen e integren los proyectos de compresión MVC y visualización 3D.

El resto del documento está organizado como se describe a continuación. Primero se hace un resumen de conceptos básicos relacionados con el proyecto: vídeo, codificación de imágenes, cómo se logra la visión tridimensional, material utilizado, conceptos sobre transmisiones en red en tiempo real sin comprimir y transmisiones estereoscópicas. Se sigue con el segundo capítulo, donde se exponen con profundidad los bloques principales del proyecto (adquisición, distribución y reproducción), junto al software desarrollado para cada uno y todas sus características. El tercer capítulo expone la evolución del proceso de implementación de cada bloque, razonando cada cambio y valorando cada progreso con las pruebas realizadas, hasta lograr el rendimiento deseado. En este capítulo también se analiza una transmisión a nivel de red, para mostrar cómo funcionan los protocolos de transmisión de tiempo real utilizados y como se deben interpretar. Para

finalizar, el cuarto capítulo presenta las conclusiones extraídas durante la realización del proyecto y las líneas futuras que se podrían seguir, o que se están siguiendo por otros alumnos, para ampliar este proyecto.

El código y la documentación del proyecto se pueden encontrar en Google Code con licencia GNU GPL v3, excepto el software de control de la cámara Videre porque su licencia no lo permite:

```
http://code.google.com/p/tfc-transmisiones3d/
```

CAPÍTULO 1. BASES TEÓRICAS

Este capítulo describe conceptos y tecnologías presentes a lo largo del proyecto. Para entender el resto del informe es importante tener claros los conceptos que se incluyen sobre vídeo y transmisiones en tiempo real.

1.1 Vídeo

La señal de vídeo es una secuencia de imágenes correlativas, reproducidas a velocidad suficiente como para obtener sensación de movimiento. Este efecto se crea cuando el cerebro es incapaz de tomarlas como imágenes independientes y las mezcla. Esto supone que al hablar de "vídeo", no se está hablando de otra cosa que de muchas imágenes con un orden fijo.

1.1.1 Codificación de imágenes sin comprimir

Este proyecto no considera la opción de comprimir el vídeo. Esto supone que los flujos de vídeo se han de tratar como imágenes individuales y sin compresión. Una imagen sin comprimir se codifica siguiendo un patrón por pixel. En este proyecto se utilizan tres: RGB, RGBA y YUV.

1.1.1.1 RGB (Red Green Blue)

RGB es un modelo de codificación de color con tres componentes por imagen. Cada componente corresponde a un color primario: rojo, verde o azul. Con la mezcla de estos se consigue cualquier otro color. En la figura 1.1 se observa una representación de las mezclas básicas.

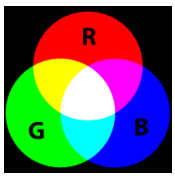


Figura 1.1 Modelo aditivo de colores rojo, verde y azul.

A cada color se le asigna un valor para indicar su proporción en la mezcla. En digital se suele destinar un byte por color para cada pixel, así que puede ir de 0 a 255. Está diseñado para que el negro, o ausencia de color, se consiga con todos los valores a 0. A su vez, el blanco se consigue con todos los valores a 255.

Los monitores de ordenador utilizan este modelo para mostrar las imágenes en pantalla. Como el cristal de las pantallas es negro, la ausencia de color se ve representada por el color original del monitor. Si el cristal fuese blanco, como en el caso de un papel al imprimir, este modelo no sería posible.

1.1.1.2 RGBA (Red Green Blue Alpha)

RGBA es un modelo de codificación de color basado en el RGB. En este caso, se le añade un byte más a cada pixel para indicar su nivel de transparencia. El campo alfa va de 0 a 255 como los demás, siendo 0 el máximo nivel de transparencia.

1.1.1.3 YUV 4:2:2

YUV es un modelo de codificación de color adaptado al ojo humano. Codifica cada imagen como un conjunto de tres componentes: una componente de luminancia y dos de crominancia. En la figura 1.2 se muestra una imagen separada por componentes. También aparece la imagen creada a partir de las 3 partes. El modelo YUV también se conoce como YCbCr, en su versión digitalizada.

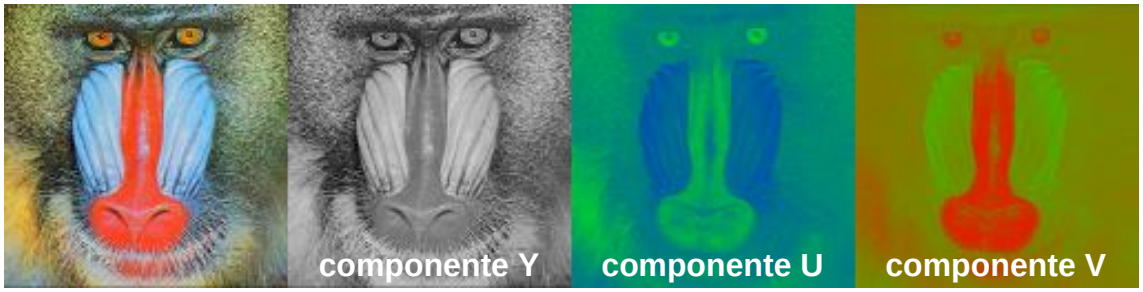


Figura 1.2 Imagen y su descomposición YUV

Cada componente tiene un uso concreto:

- La componente Y es la luminancia. Corresponde a la cantidad de luz de cada punto, y es la componente de "blanco y negro".

- La componente U es la crominancia azul (Cb). Corresponde a la cantidad de color azul de cada punto. En las imágenes de la figura 1.2 se ve muy claro, ya que los valores más altos se centran en la zona azul de la nariz del mono.

- La componente V es la crominancia roja (Cr). Corresponde a la cantidad de color rojo de cada punto. Al igual que en el caso de la componente U, aquí también se ve muy claro. Los valores más altos se encuentran donde el rojo es más presente en la imagen.

El modelo YUV se creó para ser utilizado en televisión. Cuando la industria tuvo que emitir en color se encontraron con un problema. Todas las televisiones eran compatibles con la señal en blanco y negro, y si lo cambiaban por un modelo RGB dejarían de funcionar. Una solución fue incorporar las dos señales de crominancia, como suplemento para obtener la imagen en color original. Por

esto se decidió añadir a la señal en blanco y negro dos señales más. La señal en blanco y negro quedó como señal de luminancia, y las nuevas serían la crominancia azul y la roja.

Las tres componentes se diseñaron para poder reconstruir la señal RGB mediante unos coeficientes. En la figura 1.3 se muestran las matrices de conversión.

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Figura 1.3 Matrices de reconversión YUV ↔ RGB

YUV se creó para utilizarse en sistemas analógicos, por eso estas matrices no están adaptadas a los valores binarios de 8 bits por componente y pixel que obtienen en un sistema digital. Si se utilizan estas matrices en un sistema digital, el resultado no es válido. Para realizar las reconversiones en el proyecto se han utilizado las expresiones 1.1 y 1.2. Están mejor adaptadas y dan un buen resultado al comparar el original con la reconversión [I2].

```
RGB -> YUV

Y  =  (0.257 * R) + (0.504 * G) + (0.098 * B) + 16
U  = -(0.148 * R) - (0.291 * G) + (0.439 * B) + 128
V  =  (0.439 * R) - (0.368 * B) - (0.071 * B) + 128
```

(1.1)

```
YUV -> RGB

R = 1.164(Y - 16) + 1.596(V - 128)
G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)
B = 1.164(Y - 16)                  + 2.018(U - 128)
```

(1.2)

Una de las virtudes del YUV es la posibilidad de reducir el tamaño de las componentes de crominancia. Así se permite reducir el tamaño de la imagen compuesta a costa de perder parte de la información de color, que el ojo no percibe con tanto detalle como la luminancia. En el proyecto se hace uso del submuestreo 4:2:2, que indica que por cada 4 píxeles de luminancia hay 2 de cada crominancia, es decir, los píxeles de color son el doble de grandes que los de luminancia, tal como se describe en la figura 1.4.

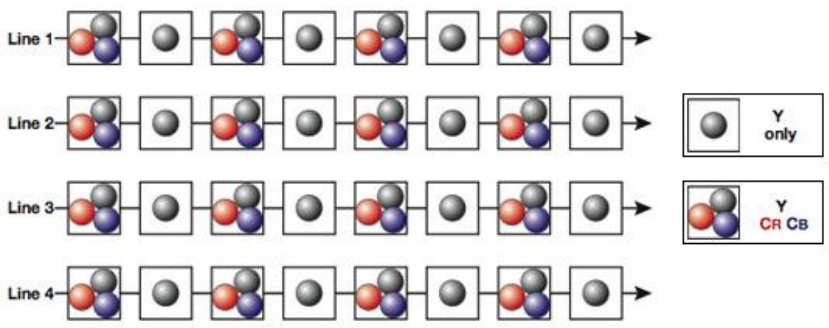


Figura 1.4 Muestreo por píxel en YUV 4:2:2

1.1.2 La visión humana

La vista es, si cabe, el sentido más importante y más utilizado por el ser humano. Incluso a la hora de realizar acciones que no la requieran, como hablar con una persona, el hecho de poder utilizarla ayuda en la comunicación.

La corteza visual del cerebro está especializada en captar imágenes y analizarlas para reconocer la escena. Se dedica a buscar elementos reconocibles y situarlos en un escenario tridimensional. Esta tarea la realiza comparando las dos imágenes que le llegan de los ojos, aunque en ocasiones no es suficiente y tiene que utilizar la perspectiva u otras técnicas ópticas.

1.1.2.1 Visión binocular

También conocida como visión estereoscópica, la visión binocular responde a la facultad de poder interpretar dos imágenes como parte de un mismo escenario, sacando de estas la información necesaria para representar un escenario en tres dimensiones. En la figura 1.5 se muestra una representación de cómo es visto por cada ojo un escenario tridimensional.

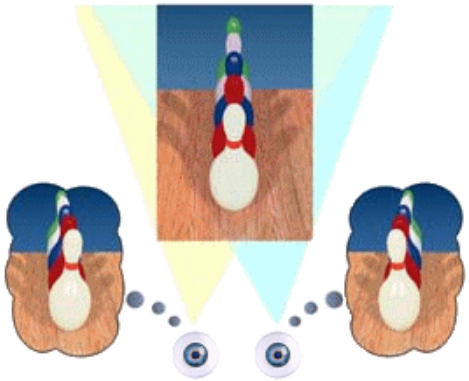


Figura 1.5 Visión binocular

La visión binocular es posible en los seres vivos que tienen órbitas frontales, como los seres humanos o las aves de presa. Esto es así porque ambas imágenes deben estar tomadas desde posiciones ligeramente diferentes. Además, ambas imágenes deben ser relacionables. De no ser así, el cerebro no las podría comparar, y si por algún problema ocular una de ellas se viese afectada el efecto de tridimensionalidad se vería comprometido.

Para comprender estas limitaciones solo hay que figurarse que imágenes le llegarían al cerebro en caso de no cumplir los requisitos, lo difícil que sería para el cerebro conseguir la información que le falta, y traducirlas en un escenario tridimensional. Al mirar la escena de la figura 1.5 uno se puede hacer una idea de qué ocurriría si una de las dos vistas no fuese correcta.

En caso de que las imágenes captadas no sean ideales, el cerebro puede ser capaz de arreglarlas y salvar el efecto tridimensional.

1.1.2.2 Perspectiva

La perspectiva es un recurso que tiene el cerebro cuando necesita más información para interpretar una imagen. Consiste en utilizar la memoria visual y relacionar deformidades de los objetos de una imagen para intentar reconstruirlos y colocarlos en un espacio tridimensional.

Observando la figura 1.6, la perspectiva es la causante de hacernos ver una calle que se pierde en el horizonte con edificios a lo largo, y no un triángulo que cruza 3/4 de la imagen con edificios flotando alrededor.

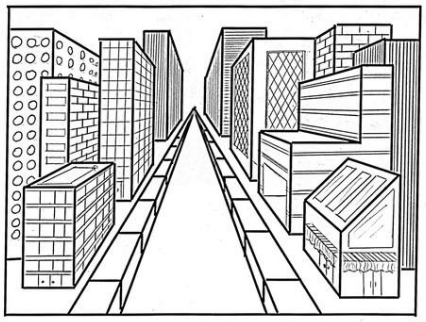


Figura 1.6 Ciudad en perspectiva

La perspectiva es un efecto muy recurrido en arte porque, en un lienzo, solo hay dos dimensiones, pero como vemos puede dar lugar a malas interpretaciones. Por suerte, el hecho de tener más de un ojo, evita que el cerebro tenga que basarse sólo en una fuente para interpretar lo que vemos.

1.1.2.3 Sensación de profundidad

El espacio es tridimensional. Esto significa que para captar correctamente el espacio es necesario conocer las tres direcciones básicas (x, y, z). La sensación de profundidad se consigue cuando el cerebro puede interpretar las imágenes que le llegan y distribuir sus partes en esas tres direcciones.

Hay varias claves que permiten percibir la profundidad, se agrupan en dos subgrupos:

Claves monoculares: percibidas por cada ojo individualmente.

- Perspectiva: tratada en el punto 1.1.2.2., consiste en calcular la distancia observando la dirección de las líneas de objetos comunes y cómo se van juntando con la distancia.

- Tamaño relativo: los objetos más distantes aparecen más pequeños frente a otros más cercanos pero de similar talla.

- Claridad de detalles: se observan más detalles en objetos cercanos.

- Interposición: un objeto que tapa parcialmente a otro se da por supuesto que está por delante.

- Sombras y tonos: la relación con la fuente de luz y las sombras dan muchas pistas sobre la colocación de un objeto.

Claves estereoscópicas: percibidas combinando los dos ojos. Están directamente relacionadas con la visión binocular del punto 1.1.2.1.

- Disparidad retinal: consiste en analizar la diferencia de ángulo según como una imagen entra por cada ojo.

- Convergencia: clave ofrecida por el mecanismo de enfoque del ojo. Según se enfoca más cerca o más lejos para ver mejor cada objeto, se consigue una pista de su situación.

Ambos grupos son importantes a la hora de detectar la profundidad. Las claves monoculares ofrecen muchos recursos de fácil aplicación, pero se pueden ver alteradas por escenarios preparados para confundir. Por su parte, las claves estereoscópicas son más complejas y fiables.

Para reproducir el efecto de profundidad en un plano en dos dimensiones, como lo sería un lienzo de un cuadro, solo se puede recurrir a las claves monoculares. Para utilizar técnicas estereoscópicas y conseguir engañar totalmente al cerebro hacen falta técnicas visuales más complejas.

Figura 1.7 R&J Beck, Mirror Stereoscope, extraída de [V4]

En la figura 1.7 se muestra un aparato de 1859 que se utilizaba para ver imágenes en tres dimensiones. Se conseguía engañar a la vista colocando una imagen para cada ojo. Con este sistema, aparte de poder percibir los efectos de profundidad monoculares típicos de una imagen, también se podían percibir los efectos estereoscópicos.

Desde que salió el aparato en forma de gafas enormes de la figura 1.7 la tecnología ha avanzado mucho, aunque aun, la gran mayoría de soluciones, suponen tener que utilizar unas gafas.

1.1.3 Técnicas modernas de visualización 3D

En el apartado 1.1.2 se han tratado varias de las peculiaridades que tiene la vista a la hora de captar imágenes, detectar posiciones y recolocar todos los elementos en un escenario tridimensional.

Actualmente la técnica ha avanzado mucho. Existen varios sistemas para engañar a la vista y conseguir ver vídeos en tres dimensiones, pero ninguno ha conseguido satisfacer la demanda de los consumidores. La vista humana está demostrando ser muy compleja, lo que está dificultando conseguir un sistema que ofrezca buenos resultados y sin un precio desorbitado.

Las técnicas modernas se centran en reproducir el contenido que se graba desde cámaras estereoscópicas como la de la figura 1.8. Estas cámaras están dotadas de dos objetivos iguales, colocados en paralelo en un plano horizontal. Cada objetivo va destinado, dependiendo su ubicación, a captar las imágenes de un ojo en concreto.

Figura 1.8 Cámara 3D HD Panasonic

Las cámaras estereoscópicas de dos objetivos consiguen un flujo de vídeo independiente para cada ojo. Estos flujos están sincronizados imagen a imagen para poderlos presentar sin problemas de desfase entre ellos. Al tener los flujos preparados, el reproductor solo tiene que dedicarse a que el usuario reciba en cada ojo el correspondiente.

Para lograr que cada ojo reciba el flujo que se ha grabado destinado para él hay diversas técnicas. Las técnicas actuales se dividen en dos grupos: con gafas y sin gafas.

1.1.3.1 Con gafas

Estas técnicas se basan en preparar la reproducción y utilizar unas gafas para que cada lente filtre el flujo que no le corresponde, dejando pasar el que sí. Existen tres variantes:

Gafas anaglíficas: Fueron las primeras gafas que se crearon para ver en 3D. Utilizan lentes de un color diferente por ojo, comúnmente rojo y azul. El vídeo se tiene que preparar filtrando cada uno, de manera que mediante las lentes coloreadas solo se pueda ver el que pertenece a ese ojo. La figura 1.9 ofrece una representación de todo el sistema.

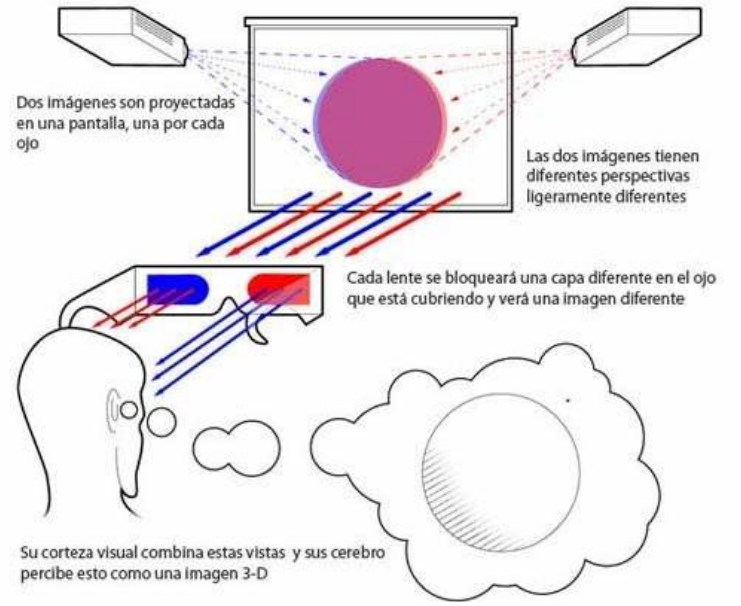


Figura 1.9 Sistema con gafas anaglíficas, extraída de [V5]

El sistema es económico, el efecto tridimensional es moderado pero los colores se ven afectados. Puede funcionar con imágenes en soporte impreso.

Gafas polarizadas: Las lentes tienen filtros de luz polarizados, girados 90º entre ellas. El reproductor ha de ser capaz de mostrar, al mismo tiempo, los dos flujos polarizados según el ojo que tenga que recibirlo. En la figura 1.10 se muestra un esquema de cómo funciona.

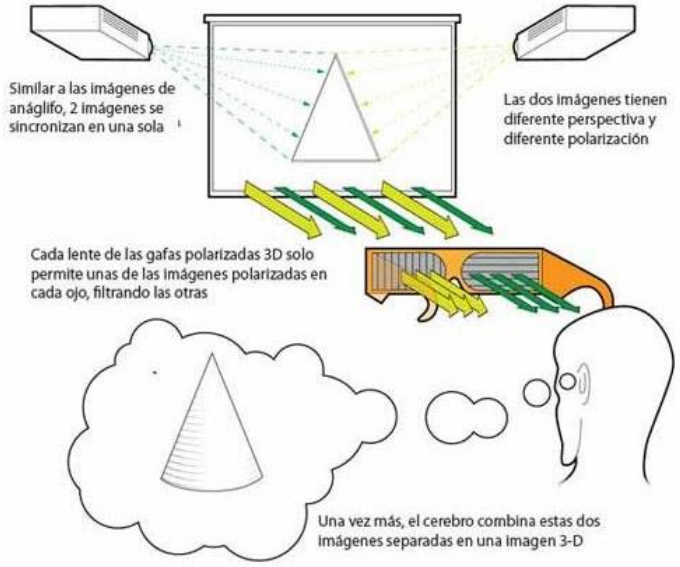


Figura 1.10 Sistema con gafas polarizadas, extraída de [V5]

El resultado es bueno pero se pierde intensidad de color al filtrar la luz, y dependiendo de la posición de las gafas puede que no filtren todo.

Gafas activas: Se sincronizan con el reproductor mediante un puerto infrarrojo. El reproductor muestra las imágenes alternando una para cada ojo. Utilizan lentes LCD que, según la señal infrarroja del reproductor, se oscurecen para dejar pasar la imagen que toque en cada momento al ojo indicado.

Actualmente este sistema es el más popular. El resultado es bueno y se utilizan monitores que permiten 100/120Hz de refresco para poder conseguir los 50/60Hz para cada ojo habituales en televisión, cosa que a priori es técnicamente factible hoy en día. Sus pegas son que se pierde intensidad de color y las gafas son más complejas de fabricar.

Uno de los proyectistas de la EETAC está realizando un reproductor de vídeo utilizando el sistema de nVIDIA con gafas activas GeForce 3D Vision [R7]. La idea es substituir el módulo de reproducción de este proyecto por el suyo, cuando esté terminado.

1.1.3.2 Sin gafas o auto-estereoscópicos

Estos sistemas están en pleno desarrollo. Funcionan con elementos bloqueantes [V7], combinados con columnas de píxeles colocados de manera que cada ojo solo vea los que corresponden a una imagen, como se representa en la figura 1.11. Existen otros métodos pero están menos avanzados [V8]. La sensación de profundidad es muy buena y no pierden tanta intensidad de color, pero los ángulos de visión son limitados. La Nintendo 3DS ha sido uno de los primeros productos en salir al mercado con esta tecnología. El ser una consola portátil para un jugador, permite que el usuario pueda centrarse en la pantalla y no se preocupe por el ángulo de vista. Recientemente ha aparecido una noticia del MIT [V6] anunciando una gran mejora en este tipo de pantallas.

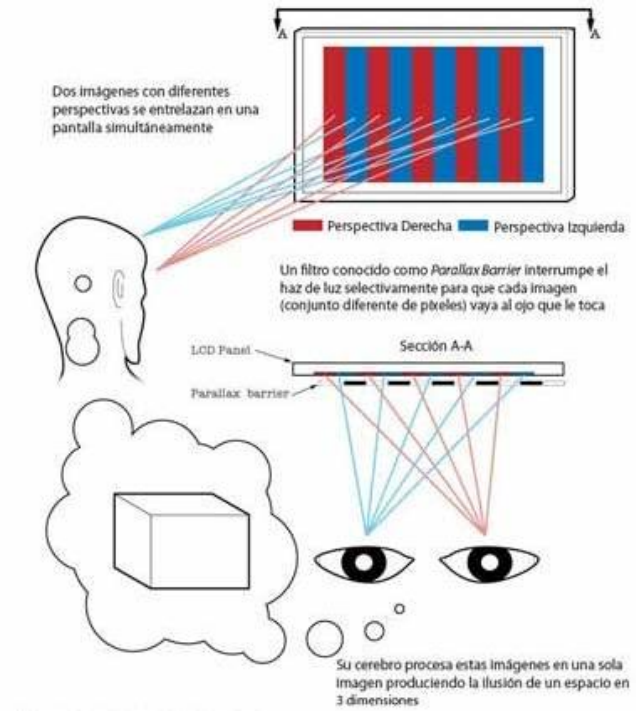


Figura 1.11 Sistema sin gafas, extraída de [V5]

1.1.4 Cámaras utilizadas

En este proyecto se han utilizado dos modelos diferentes de cámaras estereoscópicas. Tanto su interfaz como sus características son muy distintas. La más sencilla está destinada a videoconferencia en el hogar, y la otra, más profesional, al mundo de la robótica y la detección de elementos.

1.1.4.1 Minoru 3D Webcam

La cámara Minoru 3D Webcam [C1], de la empresa "Promotion and Display Techonolgy Ltd.", es la más modesta de las dos utilizadas. Está diseñada para sustituir a las webcam actuales con un solo objetivo. El propio fabricante la anuncia para utilizar con programas tipo "Skype" o "Windows Live Messenger".

Como se puede observar en la figura 1.12, incorpora dos objetivos, pero eso no le otorga capacidades estereoscópicas de serie. Viene con un pack de drivers para Windows, que la configuran y realizan el efecto estereoscópico mostrando un solo flujo de vídeo en anaglífico.

Figura 1.12 Minoru 3D Webcam

Su puerto de conexión es un USB 2.0, pero el PC la detecta como dos dispositivos diferentes. Esto es debido a que, en realidad, esta cámara con dos objetivos es en realidad dos cámaras unidas por un hub USB 2.0, y del que sale el puerto USB 2.0 que va al ordenador. El driver para Windows se encarga de hacer la unión y el efecto anaglífico, e incluso permite usar los objetivos de manera independiente. En nuestro caso, al utilizar GNU/Linux, se tienen que utilizar los drivers estándar "V4L2" [C2] (Vídeo for Linux Two) y controlar cada objetivo como si fuesen cámaras individuales.

Las capacidades técnicas se presentan en la tabla 1.1. Los 15/30 frames por segundo (fps) indicados pueden caer hasta 3/4 fps. Esto es por culpa de la velocidad de obturación, que cae si existen elementos oscuros en cualquiera de las dos imágenes. La cámara cree que es falta de iluminación y deja más tiempo de exposición, reduciendo el ritmo de captación de imágenes. Otro punto negativo es que, al ser dos cámaras unidas, no van sincronizadas y, dependiendo del PC, las imágenes entregadas pueden estar desfasadas.

Resoluciones soportadas	**640 x 480 a ~15 fps** **320 x 240 a ~30 fps**
Ajustes manuales	Enfoque independiente de cada objetivo
Tipo de imagen recibida	YUV 4:2:2 (8 bits por componente)

Tabla 1.1 Características técnicas de la Minoru 3D Webcam

1.1.4.2 Videre - STH MDCS2

"Videre Design" es una empresa dedicada al diseño y suministro de sistemas de visión y robótica [C6]. Tienen una amplia variedad de cámaras estereoscópicas destinadas a formar parte de proyectos con robots inteligentes, donde se encargan de detección y reconocimiento visual, por ello sus cámaras cuentan con características excepcionales.

El modelo utilizado en el proyecto es el que aparece en la figura 1.13. Tiene dos objetivos, un led rojo para informar de la actividad y se conecta por el puerto IEEE 1934, más conocido como puerto FireWire. Videre Design facilita un pack de drivers privados para poderla utilizar tanto en Windows como en GNU/Linux.

Figura 1.13 Videre - STH MDCS2

Al no estar destinada al uso en el hogar, los drivers no dan soporte directo a ninguna otra aplicación que no sea la suya propia. De todos modos, se entregan aplicaciones en C/C++ abiertas para poder tomar como referencia si se quiere desarrollar software para ella. Otro problema de los drivers privados es que, al ser un modelo del año 2004, dan problemas con el soporte de ordenadores de 64 bits al compilar.

Las capacidades técnicas se presentan en la tabla 1.2.

Resoluciones soportadas	**1280 x 960 a 7 fps** **640 x 480 a 25 fps** **320 x 240 a 25 fps**
Ajustes manuales	Enfoque y diafragma independiente de cada objetivo
Tipo de imagen recibida	RGBA y B&N (8 bits por componente)

Tabla 1.2 Características técnicas de la Videre - STH MDCS2

Después de probarla de manera intensa, se pueden añadir un par de virtudes más que no se pueden referenciar con datos en una tabla. Una de ellas es que las ópticas que incorpora ofrecen unas imágenes muy nítidas. La otra corresponde a su funcionamiento interno, al estar construida como un único elemento, las imágenes que entrega siempre están sincronizadas y no hay desfases entre ellas, cosa que sí puede pasar con la Minoru 3D.

1.2 Transmisión sobre una red IP en tiempo real

La principal característica de una transmisión por red en tiempo real es la necesidad de velocidad, por encima de cualquier otro aspecto. Es vital que el flujo de datos pueda transmitirse al mismo tiempo que se genera, porque si se retrasa dejaría de considerarse "tiempo real".

Esto conlleva que si un paquete se pierde es mejor descartarlo, porque si se vuelve a enviar se producen retrasos. Es mejor perder una cantidad mínima de paquetes, que produzcan algún error aceptable en destino, que no controlar las pérdidas e ir realizando esperas de paquetes reenviados. Si las pérdidas son muy severas, siempre se puede reducir la calidad de la señal, para bajar el ancho de banda y las pérdidas.

El paralelismo más claro es compararlo con una llamada de teléfono. La red ha de permitir que la voz se transmita sin esperas, porque de lo contrario no sería posible la comunicación en tiempo real. Es mejor perder algún dato y que en algún momento se oiga algún ruido, que no provocar esperas incómodas como si fuese un walkie-talkie.

1.2.1 UDP/IP

El protocolo UDP (User Datagram Protocol) es uno de los dos principales protocolos de transporte que van sobre IP (Internet Protocol). El otro es TCP (Transmission Control Protocol), pero su funcionamiento está enfocado a ofrecer una conexión fiable. Esto significa que, mientras UDP no se encarga de corregir datos corruptos durante la transmisión, ni de controlar si se pierden datos, TCP se asegura de que todos los datos hayan llegado bien, y si falta alguno, realiza una retransmisión. De este modo, un sistema TCP puede generar retrasos al esperar reenvíos de paquetes perdidos o corruptos.

Teniendo en cuenta el apartado 1.2, UDP es el más adecuado para el tipo de necesidades de este proyecto. Estas son sus características básicas:

- Requiere 8 bytes de cabecera que, comparando los 20 bytes de TCP, ofrece mayor eficiencia (mejor proporción "bytes útiles / bytes utilizados").

- Carece de mecanismos automáticos de control de pérdidas. Una vez enviado un paquete se desentiende de él.

- No necesita establecer la conexión al principio de la sesión, sólo se tiene que indicar el destino para empezar a transmitir.

- Contiene un campo de checksum en la cabecera que permite verificar la integridad de los datos, pero no corregirlos.

1.2.2 RTP

RTP (Real-time Transport Protocol) es un protocolo diseñado para ser utilizado en transmisiones en tiempo real [N1]. Funciona sobre UDP/IP, soporta vídeo entre otras opciones y tiene 12 bytes de cabecera fijos, que se añaden a partir de la cabecera UDP, más una cabecera opcional que depende de cada codificador audiovisual transportado. La razón por la que se usa RTP es que complementa a UDP, que carece de ciertos campos importantes para la transmisión en tiempo real.

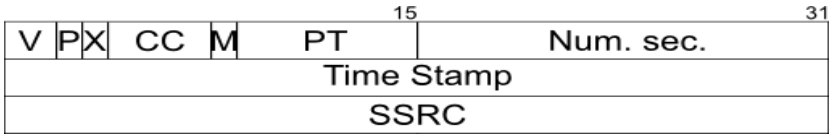


Figura 1.14 Cabecera RTP básica

La cabecera básica RTP que se añade a UDP aparece en la figura 1.14, siendo cada fila 4 bytes (32 bits) de datos. Cada campo está explicado a continuación con su tamaño concreto y el valor utilizado en el proyecto, en caso de ser estático, entre paréntesis:

- V (2 bits, 2): número de versión del protocolo RTP utilizado.
- P (1 bit, 0): padding, indica si hay relleno en el paquete.
- X (1 bit, 0): indica si utiliza alguna extensión de cabecera especial.
- CC (4 bits, 0): conteo CSRC, número de fuentes contribuyentes.
- M (1 bit): marker bit, en vídeo indica si es el último fragmento de una imagen.
- PT (7 bits): payload type, identificador del tipo de contenido.
- Núm. sec. (16 bits): número de secuencia del paquete, aumenta secuencialmente entre un paquete y otro para poderse ordenar en destino.
- Timestamp (32 bits): instante de tiempo en el que se ha generado el contenido del paquete. Todos los paquetes de una misma imagen llevan el mismo timestamp. Las unidades son ticks del reloj definido por cada codificador; en vídeo suele ser un reloj de 90 KHz, ya que es divisor del típico oscilador de 27 MHz presente en los equipos de TV, y a su vez es múltiplo de las frecuencias de imagen habituales (24 Hz, 25, 29.97, 30, 50, 59,94, 60 ...).
- SSRC (32 bits): identificador de la fuente de sincronía.

Utilizar el protocolo RTP facilita un modelo estándar de transmisión de vídeo, pero no está preparado para transmisiones que requieran un gran ancho de banda. El problema reside en el tamaño del número de secuencia, que va de 0 a 65535. Si el flujo requiere un número muy alto de paquetes por segundo, puede ocurrir que dos paquetes que estén poco separados en el tiempo tengan el mismo número de secuencia, lo cual puede ser problemático a la hora de reconstruir la imagen, si hay pérdidas o reordenaciones.

Teniendo en cuenta un flujo de 240 Mbit/s o unos 30 MBytes/s, como los de este proyecto, correspondiente a unos 24000 paquetes por segundo, el número de secuencia tardaría solo 2.73 segundos en repetirse (ecuación 1.3).

1.2.3 RFC 4175

El RFC 4175 es un estándar de paquetización para vídeo sobre RTP sin comprimir. El nombre oficial es "RTP Payload Format for Uncompressed Vídeo" [N4].

Un paquete puede contener información de más de una línea de la misma imagen, pero nunca de imágenes diferentes. Por otro lado, una línea se puede fragmentar y ser transportada en diversos paquetes.

La cabecera es de 8 bytes, pero si se incluye más de una línea en el paquete hay que sumarle 6 bytes más por línea extra. En la figura 1.15 se muestran en color los campos que se repiten por cada línea. De haber más de una línea en el paquete, primero se colocan todas la cabeceras seguidas y luego los datos de las líneas, en el mismo orden que sus cabeceras.

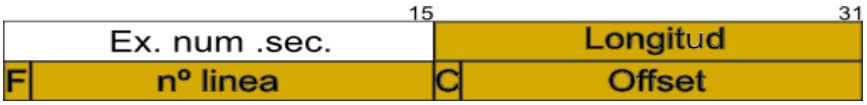


Figura 1.15 Cabecera específica RFC 4175

Cada campo está explicado a continuación con su tamaño concreto y el valor utilizado en el proyecto, en caso de ser estático, entre paréntesis:

- Ex. núm. sec. (16 bits): extensión del número de secuencia. Contiene los bits de mayor peso de un número de secuencia de 32 bits, quedando para el campo de la cabecera RTP los bits de menor peso.

- Longitud (16 bits): cantidad de bytes de la línea incluidos en el bloque de datos del paquete.

- F (1 bits, 0): indica si el vídeo está entrelazado.

- nº línea (15 bits): número de la línea que contiene respecto a la imagen a la que pertenece. Empieza en el 0.

- C (1 bit): indica si hay, mínimo, otra cabecera de fila a continuación de esta.

- Offset (15 bits): número de posición del primer pixel de la línea contenido en el paquete. Empieza en el 0.

El campo de extensión de número de secuencia soluciona el problema comentado en el apartado 1.2.2. Añade 16 bits al número de secuencia y pasa a tener valores desde 0 a $2^{32} = 4{,}29 \times 10^9$. De este modo, teniendo en cuenta el mismo flujo de 24000 paquetes por segundo, se pasaría a repetir un número de secuencia cada 49 horas. En la ecuación 1.3 aparece un ejemplo.

$$\begin{aligned}&\text{Tiempo en dar la vuelta al contador con 24000 paquetes/segundo}\\&\text{16 bits} \rightarrow 2^{16}/24000 = 2.73 \text{ segundos}\\&\text{32 bits} \rightarrow 2^{32}/24000 = 49.71 \text{ horas}\end{aligned} \tag{1.3}$$

En caso de enviar información de una sola línea, la cabecera RTP seguida de la del RFC 4175 quedaría como en la figura 1.16.

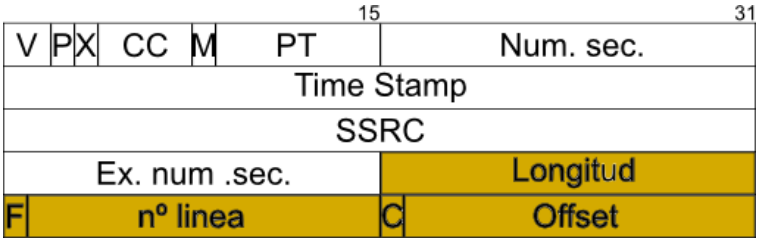


Figura 1.16 Cabecera RTP + RFC 4175

En la cabecera del RFC 4175 no aparece ningún campo para indicar el formato de vídeo. Esto es así porque se aprovecha el campo PT de la cabecera RTP. Este campo sigue el RFC 3550 para indicar qué código corresponde a cada tipo de contenido, pero es muy limitado y no tiene ninguno para el que se utiliza en el proyecto, vídeo raw YUV 4:2:2. Para estos casos, se usan los códigos del 96 al 127, asignados dinámicamente mediante una señalización SIP, RTSP o SDP previa a la transmisión RTP.

Dependiendo del formato de vídeo que se incluye en el campo de datos, el RFC 4175 incluye unas normas a seguir. Para el YUV 4:2:2 son las siguientes:

- Los componentes U y V, de forma horizontal, forman sub-grupos indivisibles a la hora de transmitir junto a las Y que les acompañan. Esto significa que la unidad mínima, e inseparable, a transmitir serán dos crominancias U y V con sus correspondientes luminancias Y. A esta unidad el RFC 4175 le da el nombre de pgroup (pixel group). En el caso de que se usen 8 bits/pixel, el grupo será de 4 bytes (1 byte U, 1 byte Y, 1 byte V, 1 byte Y).

- El orden de transmisión debe ser U_0 - Y_0 - V_0 - Y_1 - ...

- Los sub-grupos formados deben ser de píxeles adyacentes.

1.2.4 Transmisiones estereoscópicas

En el apartado 1.2.3 se habla de la transmisión de vídeo como si se tratase de un vídeo de una sola fuente, pero ese no es el caso de este proyecto. El RFC 4175 está preparado para tratar con imágenes dando por hecho que no son parte de un flujo que genera dos imágenes en cada instante, y trata todos los paquetes con un mismo timestamp como si se tratase de una misma imagen.

Para adaptar el proyecto a esta realidad surgen dos posibilidades:

- Crear dos flujos independientes.
- Fundir las imágenes en un mismo flujo (Side-by-Side).

1.2.4.1 Flujos independientes

Se crean dos flujos independientes (con puertos UDP distintos), uno por cada objetivo de la cámara, y se reordenan en destino. En la figura 1.17 aparece una representación del sistema.

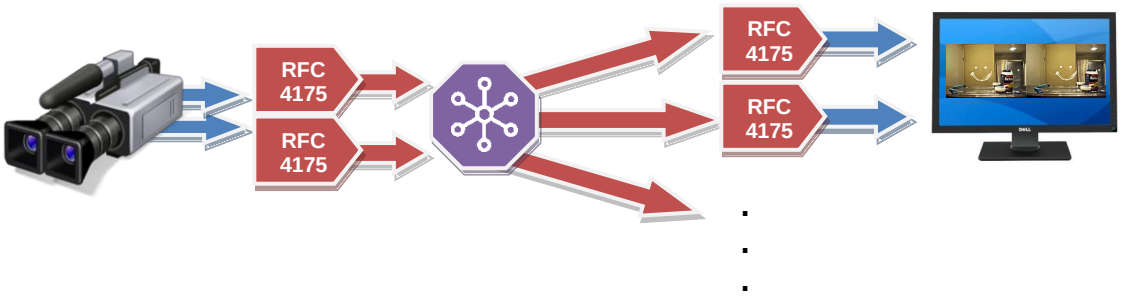


Figura 1.17 Esquema de transmisión con un flujo por vista

No es el sistema que se acostumbra a utilizar para solucionar este tipo de necesidades por varios motivos:

- Requiere realizar todos los pasos de configuración, paquetización y transmisión para cada flujo independiente.

- Requiere tener que etiquetar en origen cada flujo teniendo en cuenta la sincronización en destino.

- No facilita la sincronización en destino. El destinatario está obligado a ir buscando las parejas entre las imágenes recibidas.

1.2.4.2 Side-by-Side

Side-by-Side es un método que presenta las dos vistas como si de una sola se tratase. El sistema, una vez obtiene una pareja de imágenes de un mismo instante, las une lateralmente y las procesa como si de una misma imagen se tratase. De este modo se asegura que llegarán al reproductor sincronizadas, y no hay que realizar procesos extra para procesarlas por separado.

Este sistema acostumbra a aplicarse con un paso intermedio de reducción de tamaño. Antes de unir las imágenes, se reducen a la mitad en relación a sus columnas. Así se consigue que la imagen resultante de su unión sea del mismo tamaño que una sola de ellas.

En el proyecto se ha tenido en cuenta el sistema habitual. Igualmente, teniendo en cuenta que está pensado para redes con limitaciones con ancho de banda, y que las pruebas en el laboratorio no tienen ese problema, se ha añadido la posibilidad de realizar un "Side-by-Side extendido".

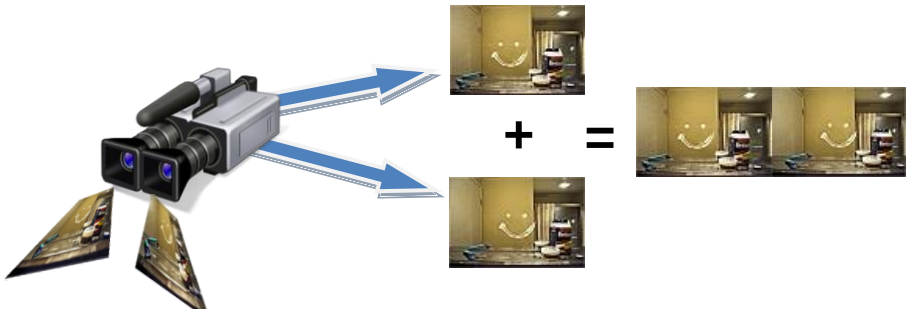


Figura 1.18 Obteniendo el Side-by-Side extendido

El Side-by-Side extendido prosigue la misma pauta del Side-by-Side, pero evita reducir el tamaño de las imágenes al unirlas. En la figura 1.18 se muestra un ejemplo de su resultado.

Side-by-Side es el sistema que se está imponiendo en las transmisiones de vídeo 3D en la actualidad (por ejemplo, las transmisiones de prueba que TV3 realiza en su canal 3HD [I6]). El estándar de la DVB para 3DTV [I7], publicado recientemente, indica que se coloca la imagen izquierda a la izquierda y la derecha a la derecha. En la figura 1.19 se muestra con claridad.

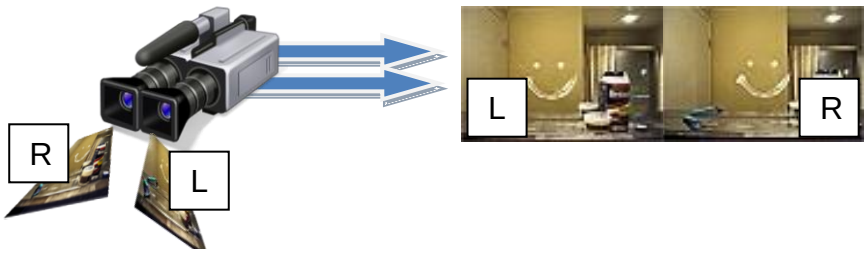


Figura 1.19 Colocación de cada imagen

CAPÍTULO 2. SOFTWARE DESARROLLADO

En este capítulo se muestran en detalle las aplicaciones que se han desarrollado para llevar a cabo el proyecto. Todo el software se ha desarrollado en C/C++ para GNU/Linux. Se pueden diferenciar en tres bloques, tal y como aparecen en la figura 2.1.

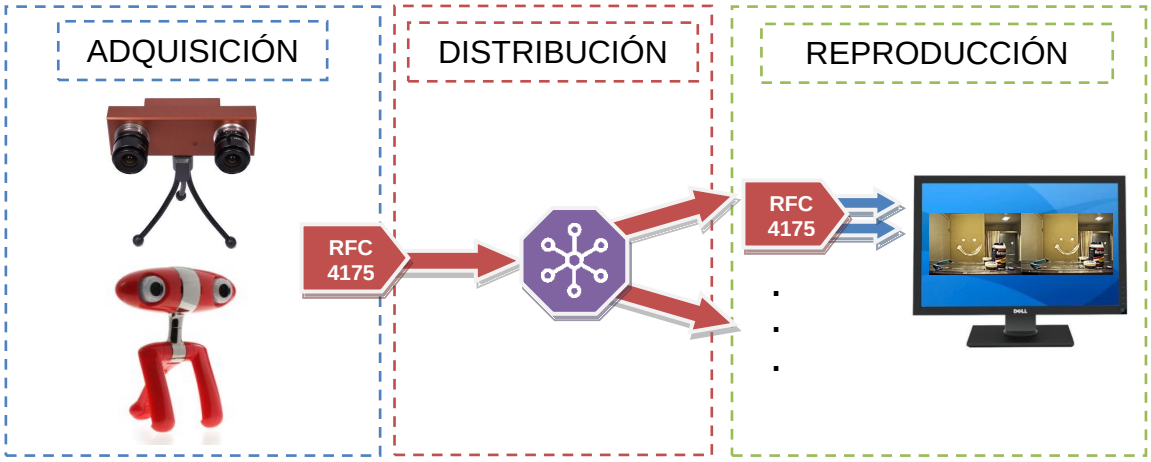


Figura 2.1 Distribución por bloques de las aplicaciones

En el bloque de adquisición las transmisiones se realizan siguiendo el RFC 4175, detallado en el apartado 1.2.3, para evitar incompatibilidades. Por ello el bloque de reproducción debe ser capaz de hacer el proceso inverso. El bloque de distribución sólo se dedica a reenviar a sus clientes del bloque de reproducción lo que le llega desde el bloque de adquisición, por lo que no es necesario que implemente el RFC 4175, y se limita a replicar paquetes RTP genéricos.

El campo "Payload Type" de la cabecera RTP se ha completado siguiendo un patrón genérico para todas las aplicaciones del proyecto. Estos son los valores que proponemos que se establezcan durante el establecimiento de la conexión con SIP, RTSP o SDP. Los códigos van del 112 al 117 siguiendo las configuraciones de la tabla 2.1.

Payload Type	Resolución	Formato
112	320 x 240	Side-by-Side extendido
113		Side-by-Side
114	640 x 480	Side-by-Side extendido
115		Side-by-Side
116	1280 x 960	Side-by-Side extendido
117		Side-by-Side

Tabla 2.1 Códigos del Payload Type asignados en este proyecto

2.1 Adquisición de vídeo estereoscópico

Para adquirir el vídeo estereoscópico, y transmitirlo por red, se han utilizado las dos cámaras analizadas en el apartado 1.1.4. Al ser tan diferentes se ha utilizado un programa diferente para cada una.

2.1.1 Adquisición con la Minoru 3D Webcam

La Minoru 3D Webcam está formada por dos webcams USB 2.0 estándar unidas a un mismo bus USB 2.0. Para realizar la adquisición se ha partido de una aplicación, escrita en C/C++, capaz de mostrar en pantalla dos webcam USB 2.0 al mismo tiempo.

2.1.1.1 Aplicación de base: v4l2stereo

La aplicación utilizada, "v4l2stereo", forma parte del proyecto "libv4l2cam" [C4] en Google Code. Este proyecto utiliza el driver genérico V4L2 para controlar las cámaras, y de OpenCV para realizar efectos y mostrar en pantalla. Al mismo tiempo, este proyecto está relacionado con el proyecto "Sentience" [C3], también en Google Code. En Sentience desarrollan aplicaciones de percepción volumétrica para robots móviles, aprovechando la visión estereoscópica con webcams. Realizan mapas de profundidad, modelos 3D a color para evitar obstáculos con fines de navegación y reconocimiento de objetos.

En el apartado de documentación de la web del proyecto Sentience aparece un completo tutorial para utilizar la aplicación v4lstereo con la Minoru 3D [C5]. Explican como desmontarla para integrarla con facilidad en un robot e incluso los pasos necesarios para importar el proyecto en Eclipse, y poder seguir desarrollando de manera fácil por tu parte. Explican que se han fijado en esta webcam porque ha sido la primera webcam 3D económica que ha salido al mercado, y que aporta una solución barata y funcional a sus necesidades.

Figura 2.2 Aspecto de la aplicación v4l2stereo

En la figura 2.2 se muestra su aspecto. Se compone por un par de ventanas a modo de reproductor del contenido ofrecido por cada objetivo.

v4l2stereo se ejecuta desde la línea de comandos y, una vez arrancada, ya no se puede reconfigurar. Acepta multitud de opciones configurables, así como seleccionar dos webcam cualesquiera como fuente de vídeo. Las opciones de configuración básicas, y sus flags correspondientes, aprovechadas para el proyecto son las siguientes:

- "`-0`" o "`--dev0`": Dispositivo de cámara izquierda.
- "`-1`" o "`--dev1`": Dispositivo de cámara derecha.
- "`-w`" o "`--width`": Configura el ancho en píxeles de las imágenes entregadas por las cámaras. Por defecto es 320.
- "`-h`" o "`--height`": Configura el alto en píxeles de las imágenes entregadas por las cámaras. Por defecto es 240.
- "`-f`" o "`--fps`": Configura las imágenes por segundo que entregarán las cámaras individualmente. Por defecto es 30.
- "`--help`": Muestra todas las opciones de configuración.

La configuración que se indique mediante los flags tiene que ser coherente con las capacidades del sistema. Esto significa que, si la Minoru 3D acepta resoluciones de 320x240 y 640x480, no se le puede indicar una resolución diferente. Lo mismo pasaría al escoger un framerate no aceptado.

```
a) #>./v4l2stereo -0 /dev/vídeo1 -1 /dev/vídeo0 -w 320 -h 240

b) #>./v4l2stereo -0 /dev/diveo1 -1 /dev/vídeo0 -w 640 -h 480 --fps 15
```

Figura 2.3 Ejecutando v4l2stereo con varias configuraciones

En la figura 2.3 se muestran dos ejemplos para arrancar la aplicación. La opción "a" configura las cámaras para una resolución de 320x240 sin indicar el framerate, que será 30 fps por defecto si no se introduce. En la opción "b" la resolución escogida es 640x480, pero como la cámara no soporta un framerate de 30 se tiene que introducir el que se desea, que en este caso son 15 fps. En ambos ejemplos los dispositivos seleccionados son los mismos y corresponden a los dispositivos que asocia GNU/Linux con cada objetivo de la Minoru 3D.

La aplicación trata las parejas de imágenes de forma secuencial. Espera a que el driver le entregue una de cada objetivo, realiza todas las opciones que se hayan indicado y luego pasa a mostrarlas en pantalla. El driver entrega las imágenes en YUV 4:2:2, a 8 bits por componente, con el orden YUYV, pero para realizar efectos y mostrarlas utiliza OpenCV. Por su parte, OpenCV necesita las imágenes encapsuladas en objetos "IplImage", que a la vez necesitan las imágenes en RGB. Así, lo primero que hace con cada pareja de imágenes es encapsular cada una en un objeto IplImage, que las pasarán a RGB. El diagrama de la figura 2.4 aclara este proceso.

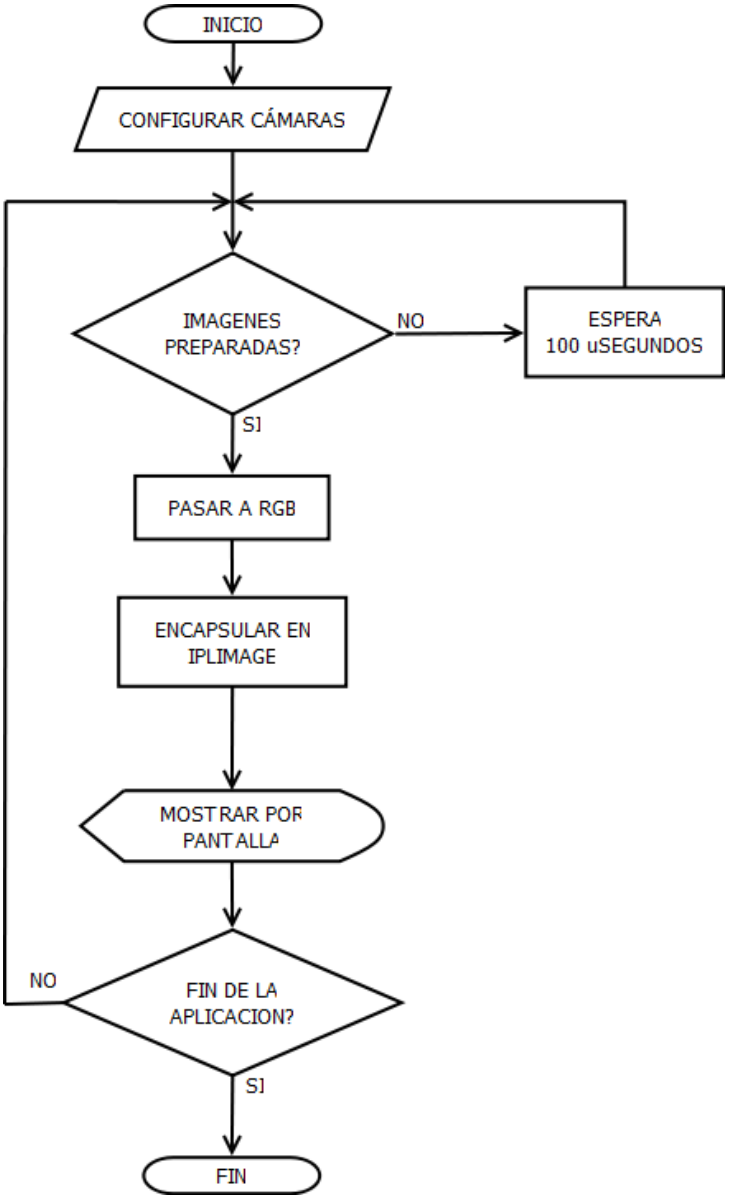


Figura 2.4 Diagrama de v4l2stereo

Suponiendo que el máximo framerate es 30, se esperan un par de imágenes nuevas cada 33 ms como máximo. Aprovechando estos datos, se permite a la aplicación realizar esperas de 100 µs entre consulta y consulta para no saturar el sistema. Hay que tener en cuenta que, sin estas esperas, la aplicación estaría realizando la consulta de "imágenes preparadas?" de manera secuencial hasta lograr salir del bucle. Este tipo de bucles descontrolados no se pueden permitir en una aplicación, porque pueden llegar a bloquear un ordenador ocupando gran parte de la potencia en esa consulta innecesaria.

La aplicación termina cuando se pulsa la tecla escape. Cuando eso ocurre, en el bloque de toma de decisiones del diagrama que indica "fin de la aplicación?" se sigue por el camino del "si".

En las primeras pruebas de la aplicación v4l2stereo, todavía sin modificarla, se apreciaba que la calidad de la imagen era muy pobre. Comparando con otras aplicaciones que utilizasen la webcam había mucha diferencia, por lo que no podía ser problema del dispositivo. Analizando el código paso a paso se encontró que el problema estaba en la conversión de YUV a RGB.

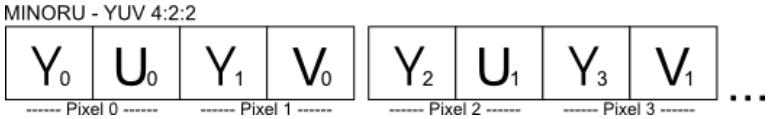


Figura 2.5 Codificación de imagen entregada por la Minoru 3D

La figura 2.5 muestra un ejemplo de cómo se recibe la imagen codificada en YUV, y en las ecuaciones 2.1 aparece como se debería realizar la conversión a RGB. El problema era que la aplicación realizaba la conversión como aparece en 2.2, descartando todas las componentes "Y" impares.

YUV 4:2:2 -> RGB

(pixel 0)
$R = 1.164(Y_0 - 16) + 1.596(V_0 - 128)$
$G = 1.164(Y_0 - 16) - 0.813(V_0 - 128) - 0.391(U_0 - 128)$
$B = 1.164(Y_0 - 16) + 2.018(U_0 - 128)$

(pixel 1)
$R = 1.164(Y_1 - 16) + 1.596(V_0 - 128)$
$G = 1.164(Y_1 - 16) - 0.813(V_0 - 128) - 0.391(U_0 - 128)$
$B = 1.164(Y_1 - 16) + 2.018(U_0 - 128)$

(2.1)

YUV 4:2:2 -> RGB

(pixel 0)
$R = 1.164(Y_0 - 16) + 1.596(V_0 - 128)$
$G = 1.164(Y_0 - 16) - 0.813(V_0 - 128) - 0.391(U_0 - 128)$
$B = 1.164(Y_0 - 16) + 2.018(U_0 - 128)$

(pixel 1)
$R = 1.164(Y_0 - 16) + 1.596(V_0 - 128)$
$G = 1.164(Y_0 - 16) - 0.813(V_0 - 128) - 0.391(U_0 - 128)$
$B = 1.164(Y_0 - 16) + 2.018(U_0 - 128)$

(2.2)

El error generaba que las imágenes perdiesen la mitad de la información de luminancia y, al ser en origen 4:2:2, que la imagen RGB estuviese formada por columnas duplicadas contiguas. Por eso al observar las imágenes daba la impresión de que la resolución fallaba.

Una vez detectado el error y encontrada su solución, se comunicó en la web del proyecto Sentience para que procedieran a arreglarlo en su código [C5, comentarios del 4 de Octubre de 2010].

2.1.1.2 v4l2stereo adaptada al proyecto

La adaptación de la aplicación a las necesidades del proyecto se ha realizado interfiriendo lo mínimo en el funcionamiento de esta, y aprovechando su estructura principal. Se han ignorado todas las funciones que no interesaban pero no se han eliminado. El diagrama de la figura 2.6 muestra como ha quedado la estructura principal de la aplicación.

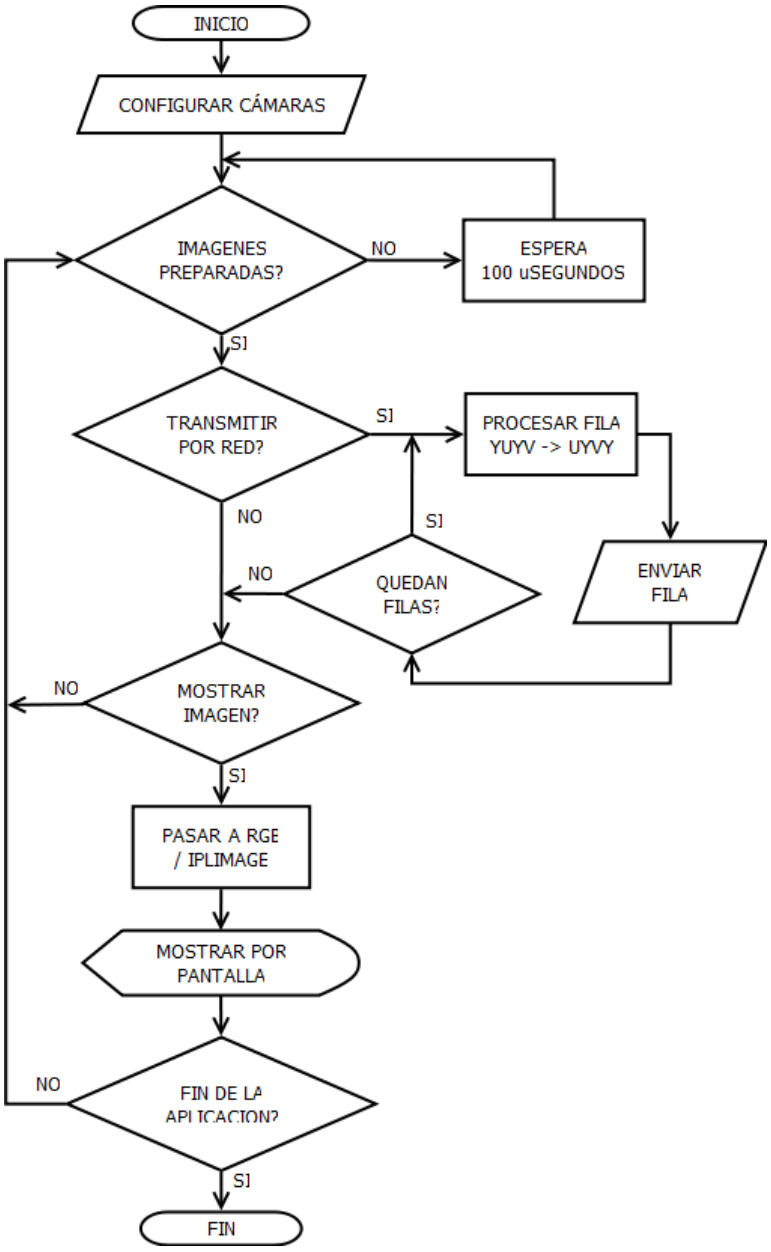


Figura 2.6 Diagrama de v4l2stereo adaptada

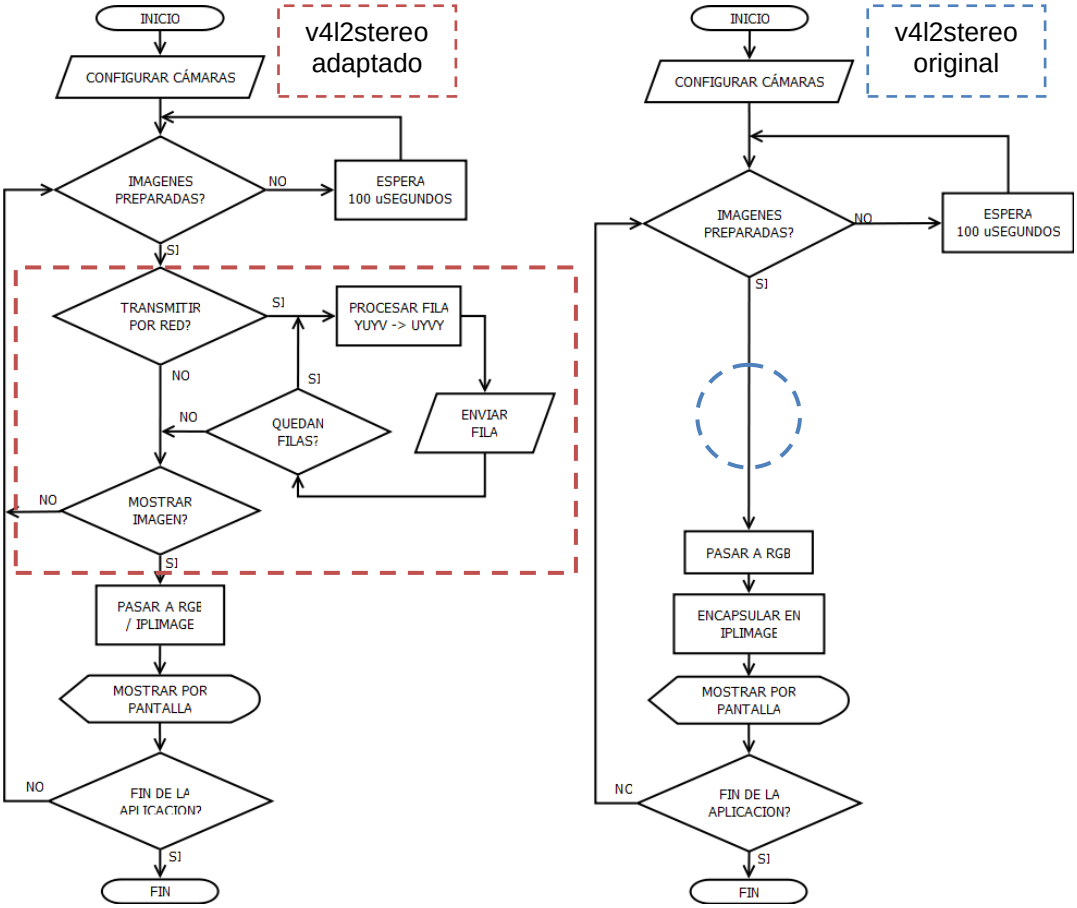


Figura 2.7 Comparación entre v4l2stereo y su adaptación al proyecto

En la figura 2.7 se pueden observar muy fácilmente los cambios realizados sobre la versión original. La lista completa de características es la siguiente:

- Se ha añadido la posibilidad de no mostrar en pantalla las imágenes y, con ello, conseguido un importante ahorro de recursos.

- Se ha añadido la posibilidad de enviar por red las imágenes fila a fila. Cada transmisión se realiza siguiendo el RFC 4175 sobre RTP, analizado en el punto 1.2.3.

- Los paquetes enviados aceptan un máximo de 1280 bytes. Las líneas que superan ese tamaño se reparten en varios paquetes.

- Se permite configurar la transmisión siguiendo el sistema "Side-by-Side" o el " Side-by-Side extendido". Ambos analizados en el punto 1.2.4.2.

- La aplicación realiza una espera auto-gestionada entre envío y envío, dependiendo del framerate para intentar evitar picos de carga en la red. También se muestra en la terminal el framerate instantáneo.

- La IP y el puerto de destino se tienen que indicar mediante flags al arrancar la aplicación, al mismo tiempo que las otras opciones.

Para estos añadidos se ha creado la siguiente serie de flags que se añaden a los que ya había:

- "`--sbsex`": Activa la transmisión de las imágenes en modo Side-by-Side extendido.
- "`--sbs`": Activa la transmisión de las imágenes en modo Side-by-Side.
- "`--nowindow`": Cancela la muestra en pantalla de las imágenes que recibe la aplicación. También evita tener que realizar los procesos de YUV a RGB y el posterior empaquetado en IplImage.
- "`--ip`": Dirección de destino del flujo transmitido. Por defecto es localhost.
- "`--puerto`": Puerto de destino del flujo transmitido. Por defecto es 25225.

Un punto muy importante del proceso de transmisión es la adaptación de la imagen al estándar RFC 4175, teniendo en cuenta el modo de imagen seleccionado para enviar. La aplicación obtiene la imagen en YUV 4:2:2 ordenada siguiendo el patrón "YUYV...". En cambio, el estándar exige que, para YUV 4:2:2, el patrón debe ser "UYVY...". Esto obliga a reordenar los datos antes de realizar la transmisión, tal y como aparece en la figura 2.8.

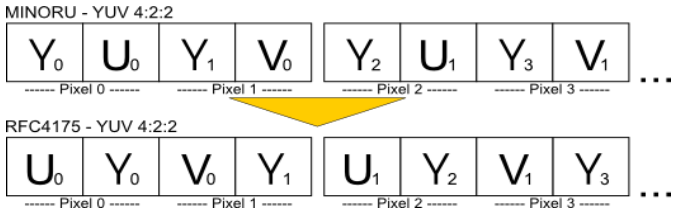


Figura 2.8 Ordenar YUV 4:2:2 YUYV a UYVY

En el caso de que esté seleccionado el modo Side-by-Side, y no el Side-by-Side extendido, la aplicación tiene que efectuar la reducción de columnas a la mitad a la vez que reordena los datos. El proceso se muestra en la figura 2.9.

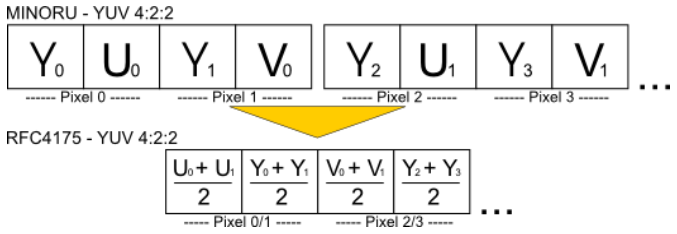


Figura 2.9 Ordenar y reducir YUV 4:2:2 YUYV a UYVY

2.1.2 Adquisición con la Videre - STH MDCS2

La Videre STH MDCS2 es una cámara destinada a desarrollo e investigación profesional. Muestra de ello es que incorpora un bus IEEE 1394 para conectarse con el ordenador, y que requiere unos drivers propietarios. Estos drivers, como ya se ha comentado en el apartado 1.1.4.2, no se integran en el sistema operativo para funcionar con otras aplicaciones, pero traen varias aplicaciones para utilizar la cámara como ejemplo de sus posibilidades.

2.1.2.1 Aplicación de base: SVS Stereo System

La aplicación escogida para utilizar de base es "SVS Stereo System". Esta aplicación es la más básica de un conjunto de aplicaciones con diferentes añadidos cada una, que no son de utilidad para las necesidades del proyecto.

Tanto los drivers como el pack de utilidades y ejemplos se han descargado de la web de Videre Design, el fabricante de la cámara. Para ello se necesita usuario y contraseña. En nuestro caso se habían extraviado y hubo que ponerse en contacto con ellos para que nos los reenviasen.

La aplicación lleva el nombre de "smallv" y se encuentra en la carpeta "samples", pero antes de arrancarla hay que configurar el path con el siguiente comando:

```
#> export LD_LIBRARY_PATH=/home/nombre_usuario/../svs44g/bin:$LD_LIBRARY_PATH
```

En ".." se debe incluir la dirección de la carpeta donde se extrajo el paquete de drivers y ejemplos.

La aplicación smallv tiene el aspecto que muestra en la figura 2.10. Como se puede observar, ofrece una interface gráfica con muchas opciones. Gran parte de ellas se describen en el documento del anexo E.

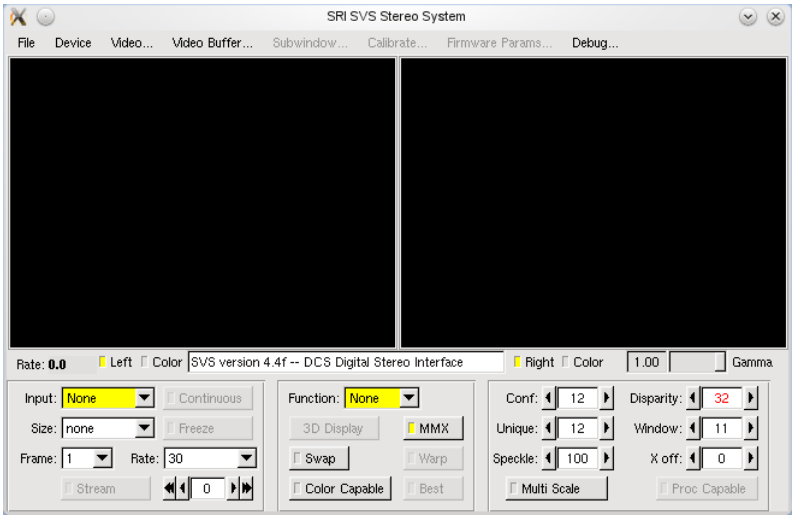


Figura 2.10 SVS Stereo System

De todos los elementos que aparecen en la interface gráfica, los importantes son los siguientes:

- Device: es un menú de la barra de menús que muestra los dispositivos conectados al ordenador que acepta la aplicación. Aquí debe aparecer la cámara seleccionada.

- Input: es un desplegable que permite escoger la fuente de extracción del vídeo. Para seleccionar la cámara se ha de marcar la opción "Vídeo".

- Size: es un desplegable que muestra varios tipos de resolución de imagen. Se ha de seleccionar "320x240", "640x480" o "1280x960", que son las que acepta la cámara.

- Rate (desplegable): es un desplegable que muestra varias frecuencias. Se ha de seleccionar una aceptada por la cámara, según la resolución escogida con anterioridad. Hay que tener en cuenta que siempre se podrá escoger una inferior al aceptado a cierta resolución, pero nunca superior. Se ha observado que a 30Hz las imágenes van a 25fps.

- Rate (indicador): es un indicador de texto que muestra el framerate al que está mostrando las imágenes.

- Continuous: es un botón que activa la recepción de imágenes según los parámetros escogidos. Las imágenes se verán en los marcos que ofrece la ventana, que en la figura 2.10 aparecen en negro.

- Color: es un botón, debajo de cada marco de imagen, que activa la recepción en color de esa imagen.

- Gamma: es un "slider" que modifica la gama de iluminación de las imágenes mostradas en los marcos.

La aplicación recibe las imágenes en color en RGBA con 8 bits por componente, tal como se muestra en la figura 2.11, cuando está activada la función. Mientras tanto solo se reciben en escala de grises a 8 bits por pixel.

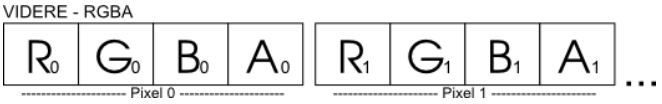


Figura 2.11 Codificación de imagen entregada por la Videre

Haciendo comprobaciones se detectó que, en las recepciones en color RGBA, la componente "A" siempre está a cero. Nunca varía su valor. Cosa en parte lógica porque una cámara no puede detectar transparencias como tal. En todo caso, ignoramos su valor.

2.1.2.2 SVS Stereo System adaptada al proyecto

Al adaptar la aplicación se ha ido con mucho cuidado, ya que de serie es muy compleja y no se quería interferir en su funcionamiento. Para manipularla se ha utilizado un sistema operativo en 32 bits (Open Suse 11.3) en máquinas de 64 bits, ya que el driver no compilaba en 64 bits. Para ejecutar la aplicación no existe este problema y corre tanto sobre 32 como 64 bits.

Para extraer las imágenes se aprovecha el sistema que las muestra en pantalla. En el momento que la aplicación recibe la pareja de imágenes en color, si está seleccionada la opción de enviar, pasan a enviarse a la dirección IP de destino siguiendo el RFC 4175 sobre RTP y el modo seleccionado.

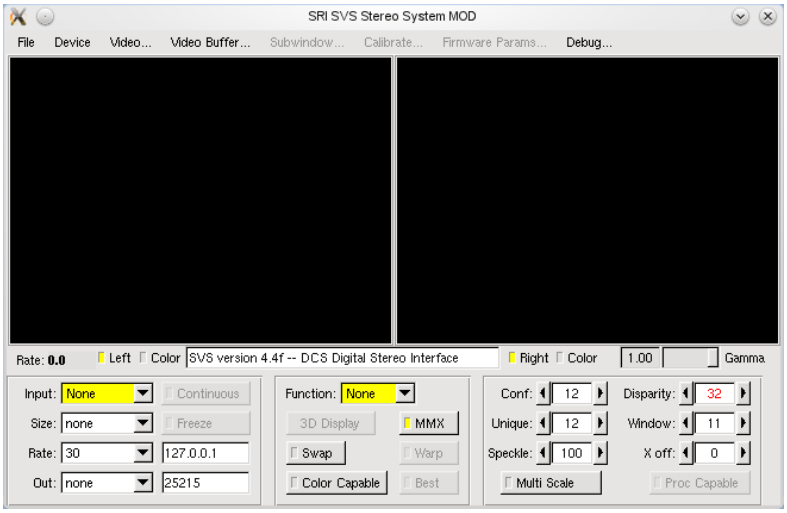


Figura 2.12 SVS Stereo System modificada

En la figura 2.12 se puede ver el resultado de las modificaciones en la interface gráfica. Se ha aprovechado la distribución original para colocar los campos necesarios en el lugar de otros menos necesarios. Ya no aparecen elementos como el desplegable "Frame", el botón "Stream" o el selector numérico. Se han incluido un par de campos de texto y el desplegable "Out". Al título del marco se le ha añadido "MOD" para diferenciar la aplicación.

En la figura 2.13 se puede ver con más claridad que elementos han sido desplazados, eliminados o creados. El desplegable "Rate" se ha recolocado para darle un aspecto más regular a la aplicación.

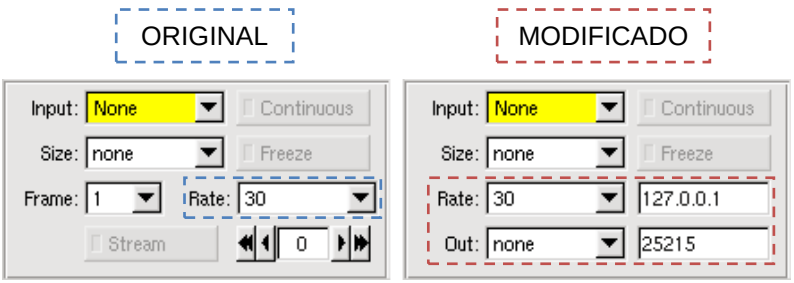


Figura 2.13 Cambios en SVS Stereo System

Los elementos añadidos se explican a continuación:

- Out: es un desplegable que permite seleccionar el modo de transmisión de las imágenes. Al momento de seleccionarlo, si por los marcos se están viendo las imágenes en color, se inicia la transmisión siguiendo los parámetros actuales. Al seleccionar "none" se cancela la transmisión.

- Campo de texto "127.0.0.1": es un campo donde deber ir la dirección IP de destino del flujo a transmitir. Por defecto es localhost.

- Campo de texto "25215": es un campo donde deber ir el puerto de destino del flujo a transmitir. Por defecto es 25215.

Las transmisiones siguen el estándar RFC 4175 para YUV 4:2:2, pero las cámaras entregan las imágenes en RGBA. Para hacer la conversión a Side-by-Side extendido se sigue el esquema de la figura 2.14.

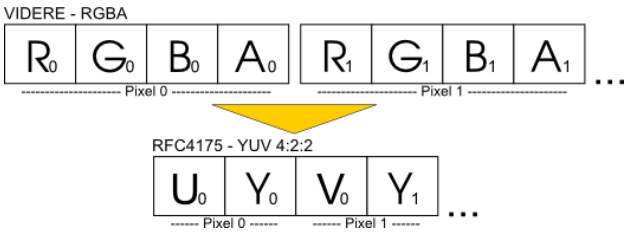


Figura 2.14 VIDERE-RGBA a YUV 4:2:2 UYVY

Se puede observar que, por las propiedades del YUV 4:2:2, las expresiones 1.1 del apartado 1.1.1.3 se deben adaptar a la reducción a la mitad de los campos de crominancia. El resultado de la adaptación son las expresiones 2.3. Estas expresiones han sido preparadas para perder la menor cantidad de información posible, realizando medias (interpolando) entre los campos que debían compartir valor. A la vez, permiten realizar las operaciones en una sola instrucción, sin tener que guardar en memoria los campos que luego se deben juntar en un mismo valor.

RGBA -> YUV 4:2:2 UYVY **(2.3)**

$$U_0 = \frac{-0.148*(\mathrm{R}_0 + \mathrm{R}_1)}{2} + \frac{-0.291*(G_0+G_1)}{2} + \frac{0.439*(B_0+B_1)}{2} + 128$$

$$Y_0 = 0.257 * R_0 + 0.504 * G_0 + 0.098 * B_0 + 16$$

$$V_0 = \frac{0.439*(\mathrm{R}_0 + \mathrm{R}_1)}{2} + \frac{-0.368*(G_0+G_1)}{2} + \frac{-0.071*(B_0+B_1)}{2} + 128$$

$$Y_1 = 0.257 * R_1 + 0.504 * G_1 + 0.098 * B_1 + 16$$

Para el modo Side-by-Side, se puede hacer realizando la media de las columnas del modo Side-by-Side extendido, pero no es lo más adecuado si se intenta ahorrar tiempo en proceso. Para realizar la conversión en el menor tiempo posible, lo recomendable es realizar todas las operaciones secuencialmente, sin guardar los valores en memoria para volver a operar con ellos. Es mejor porque entre realizar las operaciones para Side-by-Side extendido, guardar en memoria los resultados, y volver a recorrer los valores nuevos para realizar las operaciones y volver a guardar en memoria, se consumen recursos innecesarios.

Hay que tener en cuenta en estos casos que, la escritura en memoria es mucho más lenta que la realización de una operación, por lo que vale la pena complicar la operación si se pueden evitar pasos intermedios de escritura. Del mismo modo, también es importante guardar en memoria valores de operaciones complejas que no van a variar en muchos ciclos.

```
RGBA -> YUV 4:2:2 UYVY /2
```

$$U_0 = \left(\begin{matrix} \frac{-0.148*(\mathrm{R}_0+\mathrm{R}_1)}{2} + \frac{-0.291*(G_0+G_1)}{2} + \frac{0.439*(B_0+B_1)}{2} + 128 \\ + \frac{-0.148*(\mathrm{R}_2+\mathrm{R}_3)}{2} + \frac{-0.291*(G_2+G_3)}{2} + \frac{0.439*(B_2+B_3)}{2} + 128 \end{matrix} \right) \Big/ 2 \qquad \textbf{(2.4)}$$

$$Y_0 = \begin{pmatrix} 0.257*R_0 + 0.504*G_0 + 0.098*B_0 + 16 \\ +0.257*R_1 + 0.504*G_1 + 0.098*B_1 + 16 \end{pmatrix} \Big/ 2$$

$$V_0 = \left(\begin{matrix} \frac{0.439*(\mathrm{R}_0+\mathrm{R}_1)}{2} + \frac{-0.368*(G_0+G_1)}{2} + \frac{-0.071*(B_0+B_1)}{2} + 128 \\ + \frac{0.439*(\mathrm{R}_2+\mathrm{R}_3)}{2} + \frac{-0.368*(G_2+G_3)}{2} + \frac{-0.071*(B_2+B_3)}{2} + 128 \end{matrix} \right) \Big/ 2$$

$$Y_0 = \begin{pmatrix} 0.257*R_2 + 0.504*G_2 + 0.098*B_2 + 16 \\ +0.257*R_3 + 0.504*G_3 + 0.098*B_3 + 16 \end{pmatrix} \Big/ 2$$

Para este caso se han preparado las expresiones 2.4. Estas realizan todas las operaciones que necesita cada componente YUV 4:2:2, como en el caso del Side-by-Side extendido, y al mismo tiempo realizan la media de la imagen.

2.2 Distribución

El bloque de distribución es el encargado de recibir el flujo del emisor y reenviarlo entre los clientes que se hayan dado de alta. Esto nos permitirá hacer copias unicast del flujo recibido, hacia N clientes receptores. El software tiene que ser capaz de administrar estos clientes de manera dinámica, pero no tiene por qué ser compatible con el RFC 4175 (de hecho, el flujo RTP es transparente, ya que la duplicación de los paquetes se realiza a un nivel inferior, a nivel UDP) porque solo se dedica a reenviar a sus clientes todo el contenido del payload de los paquetes UDP que recibe. Para realizar la aplicación se ha aprovechado una que ya existía para facilitar el trabajo.

2.2.1 Aplicación de base: RTP Unicast Mirror

La aplicación escogida como base para crear el repetidor se llama "RTP Unicast Mirror" [N6] y está escrita en C. Su funcionamiento es muy básico, aparece en el diagrama de la figura 2.15.

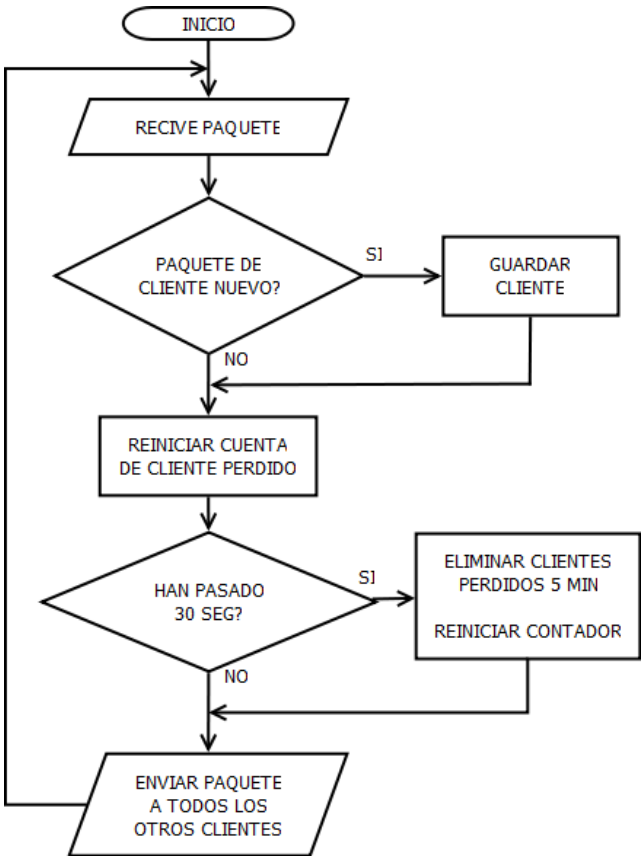


Figura 2.15 Diagrama de RTP Unicast Mirror

La aplicación propaga a toda su lista de clientes cualquier paquete que le envíe uno de estos. Cuando recibe un paquete de un cliente desconocido, lo guarda en su lista de clientes. Si lo conoce, reinicia su contador de actividad. Los paquetes deben ir sobre el protocolo UDP. La única configuración que requiere es indicar el puerto de escucha para recibir y enviar los paquetes. Esta se realiza al arrancarlo colocando el número de puerto seguido, tal que así:

```
#> ./repetidor numero_puerto
```

El contador de actividad se encarga de eliminar de la lista los clientes que llevan más de 5 minutos inactivos. El sistema considera que, si un cliente lleva 5 minutos "perdido", sin enviar ningún paquete, no debe formar parte de la lista de clientes. La aplicación permite conexiones de dos puertos mediante el tag "rtp". Esta capacidad no es útil para el proyecto así que se ha ignorado.

Según el creador, Julian Highfield, la aplicación fue concebida para realizar multidifusión de conferencias. De esta manera, si alguien quería participar en

una multiconferencia, solo tenía que empezar a transmitir hacia el servidor con la aplicación y, al momento, recibiría los flujos de los otros conferenciantes, a la vez que estos el suyo.

2.2.2 RTP Unicast Mirror adaptado al proyecto

La aplicación ofrece una buena base, pero el funcionamiento se ha tenido que cambiar en gran parte. Originalmente la aplicación repite lo que recibe, de cualquier cliente, a todos los demás. Para el proyecto, solo se deben propagar los paquetes recibidos de la fuente de vídeo.

Idealmente, el control de la sesión y la gestión de los clientes deberían realizarse con el protocolo SIP, pero por cuestiones de tiempo no ha sido posible adaptarlo a la aplicación. Para gestionar los clientes se ha desarrollado una señalización específica, conjuntamente con un protocolo de actuación para los paquetes recibidos, que es el siguiente:

- El primer paquete recibido por la aplicación se le otorga a la fuente. Esto significa que, el primer cliente que envíe un paquete, será considerado la fuente.

- La fuente no se ve afectada por el sistema de clientes inactivos. Nunca será eliminada de la lista.

- Los paquetes de la fuente nunca son explorados, solo se reenvían.

- El primer paquete de un cliente debe contener el número de puerto que desea para recibir la transmisión.

- Un paquete con el contenido "BYE" da de baja al cliente al momento.

- Los paquetes que envíen los clientes, para renovar su estatus de conectado, no son explorados. Igualmente, es preferible que incluyan el contenido "ACK".

La señalización se muestra, junto a capturas de red, en el anexo C.

La aplicación puede correr en la misma máquina que la fuente, e incluso en la misma máquina que un cliente, o de ambos a la vez. El método que decide si el cliente es nuevo comprueba solo la dirección IP, por lo que un segundo cliente desde la misma máquina no se daría de alta si utiliza la misma IP.

Hay que tener en cuenta que la fuente no cambia nunca durante la ejecución. Para seleccionar otra fuente se tiene que reiniciar el repetidor, y que, el equipo destinado a ser fuente, envíe el primer paquete que reciba este.

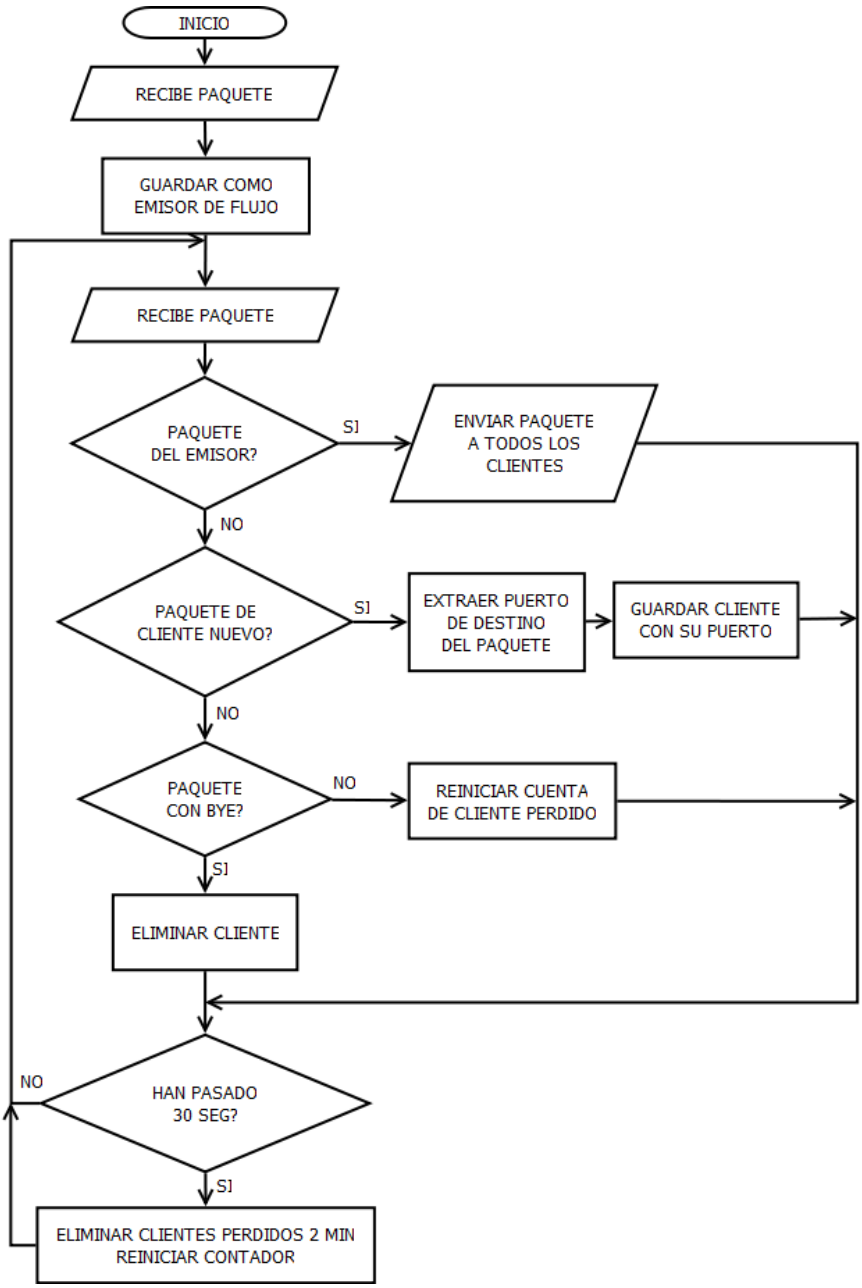


Figura 2.16 Diagrama de RTP Unicast Mirror adaptado

La figura 2.16 muestra en un diagrama como ha quedado el hilo de procesos de la aplicación, después de adaptarla a las necesidades del proyecto.

El contador de actividad se ha reajustado a dos minutos. De este modo, se considera cliente perdido a uno que lleve dos minutos sin enviar un paquete para renovar su estatus. La comprobación se hace cada 30 segundos.

Todas las transmisiones son UDP, por lo que no hay garantías de que lleguen. Se recomienda a los clientes, al menos, enviar un par de paquetes durante esos dos minutos (como un keepalive). Siguiendo esta pauta, si se perdiese uno de los paquetes no ocurriría nada, pero si se perdiesen los dos el cliente sería dado como perdido por error.

Para dejar más claro el funcionamiento del repetidor se ha realizado el ejemplo de la figura 2.17. En él aparecen dos clientes que conectan con el repetidor en dos ocasiones, tras perder la conexión la primera. La diferencia es que el cliente B la pierde porque no envía paquetes para renovar la sesión, mientras que el cliente A la pierde porque, en la primera ocasión, el paquete de actualizar sesión no llega a su destino.

La segunda desconexión del cliente B llega por el mismo motivo que la primera, en cambio, la del cliente A se realiza por petición, ya que en ese caso sí se ha realizado la actualización de sesión.

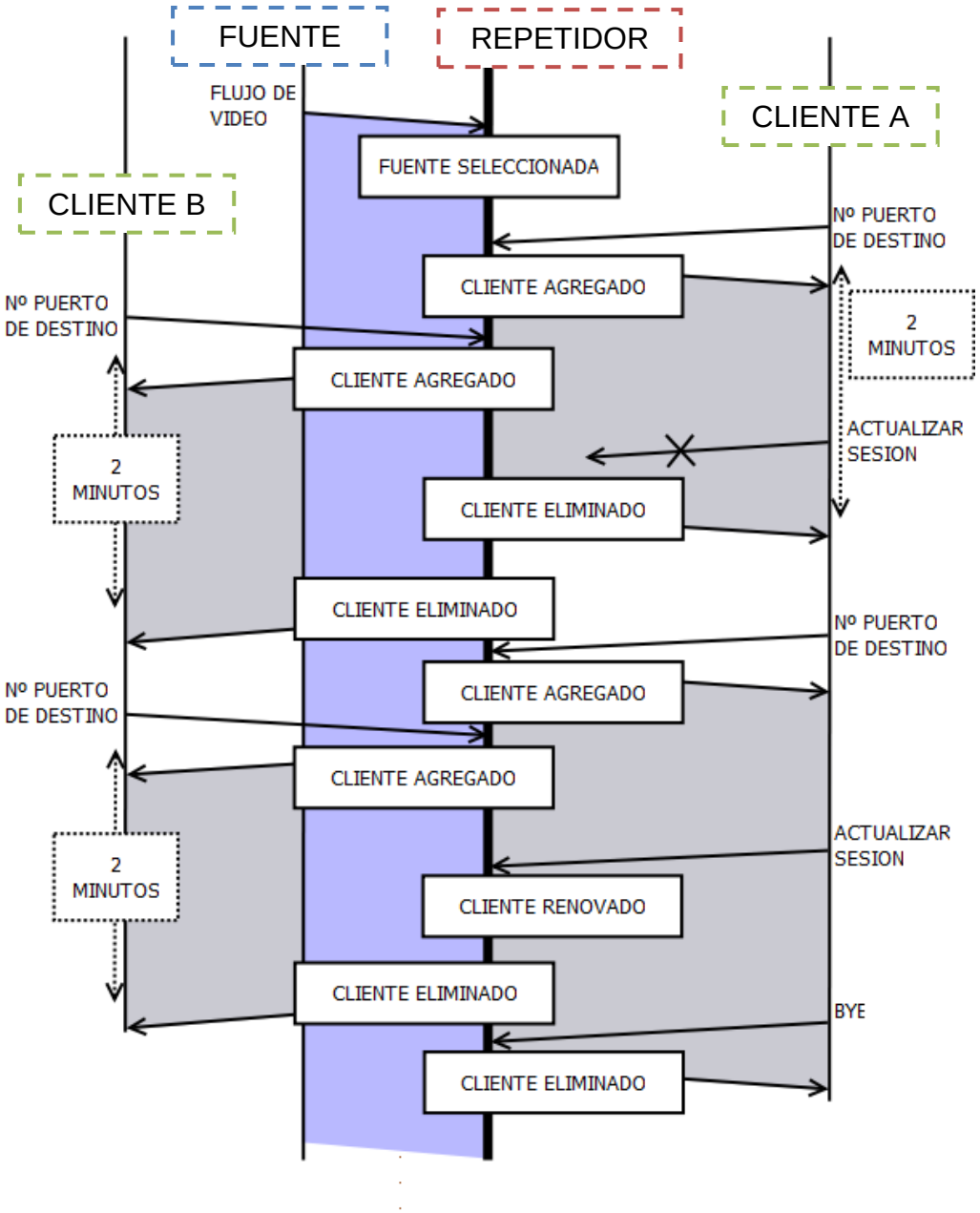


Figura 2.17 Ejemplo de inicio/fin de sesión

2.3 Reproducción

El bloque de reproducción es el encargado de pedir la transmisión de vídeo al bloque de distribución, recibirla y mostrarla en pantalla. El proceso de petición se realiza siguiendo el protocolo de actuación detallado en el punto 2.2.2, la recepción tiene que reconstruir el flujo siguiendo el RFC 4175, y la reproducción muestra las imágenes recibidas, pero sin aplicar ningún efecto tridimensional.

Para realizar el reproductor se ha partido de la aplicación v4l2stereo adaptada para el módulo de adquisición. Se ha eliminado todo el sistema de control de cámaras, efectos extra que realizaba la aplicación, se ha añadido la recepción por red y se ha aprovechado el sistema de visionado en pantalla. La figura 2.18 muestra en un diagrama como ha quedado la estructura de la aplicación que, en este caso, se ha renombrado como "DoritoTV".

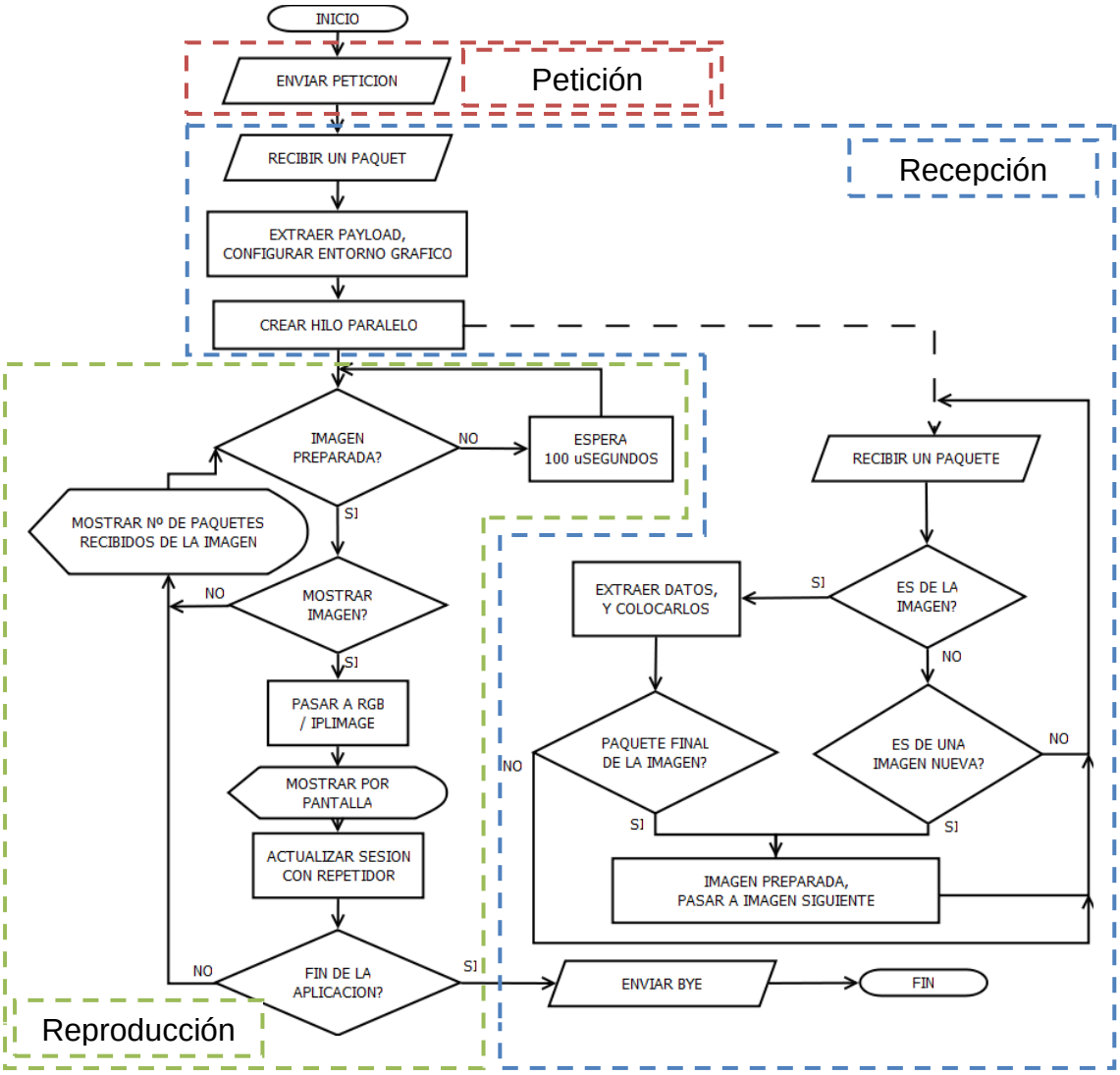


Figura 2.18 Diagrama DoritoTV

2.3.1 Petición

Al arrancar la aplicación se necesita realizar la petición al repetidor para que le transmita el flujo de vídeo. En cambio, si el escenario no incluye un repetidor, y es la fuente quien transmite directamente al reproductor, no es necesaria. En este caso se puede obviar, la aplicación la realizará por defecto sin tener ningún efecto, y el flujo recibido se reproducirá igual.

Para realizar la petición con éxito al repetidor, existen unos campos que se deben configurar al arrancar la aplicación. Se añaden a modo de flag en la terminal, siguiendo el nombre de la aplicación. Son los siguientes:

- "`--ip`": Dirección ip del repetidor. Por defecto es localhost.
- "`--pfuente`": Puerto del repetidor. Por defecto es 25555.
- "`--puerto`": Puerto de recepción para la transmisión. Es el puerto que abrirá la aplicación para recibir el vídeo, y que le entrega al repetidor para que pueda transmitirlo. Por defecto es 25225.

Un ejemplo de arrancar la aplicación, realizando la petición al repetidor en la máquina 192.168.1.15, puerto 23456, para que transmitiera en el puerto 26666 sería:

```
#> ./DoritoTV --ip 192.168.1.15 --pfuente 23456 --puerto 26666
```

2.3.2 Recepción

La recepción se inicia con la captura de un solo paquete. De este, se extrae la información del "PayLoad Type" de la cabecera RTP y, con él, se consigue la información de las propiedades del flujo. En la tabla 2.1, que se encuentra al inicio del capítulo 2, se muestra la relación entre los valores de payload y la configuración del vídeo. Con el uso de este campo, se evita tener que realizar conversaciones entre las aplicaciones, para saber el contenido del flujo. Por contra, al ser una señalización personalizada, hay que exportarla a todas las aplicaciones que formen parte del escenario. Con la recepción del primer paquete, también se aprovecha para empezar a crear la primera imagen.

La aplicación consume suficientes recursos del sistema como para requerir procesos en paralelo. Por este motivo, el módulo de recepción se deriva a otro hilo o thread. La aplicación, cuando solo tenía un hilo, perdía paquetes al utilizar el mismo para recibir y reproducir y, con ello, trozos de imagen.

Para mejorar el rendimiento de recepción/reproducción se utiliza un doble buffer de imagen. Cuando se completa una imagen en uno, se utiliza otro buffer para ir recibiendo la siguiente, mientras se muestra la actual. Con esto, se permite seguir procesando otros paquetes, sin peligro de corrupción de datos en la imagen completada.

Los dos buffers que se utilizan forman parte de contenedores "DoritoImage". Estos contenedores se han creado para la aplicación y sirven para tratar con las imágenes a nivel de paquete de red. Cuando la aplicación recibe un paquete, lo primero que hace es añadirlo al contenedor que esté activo en ese momento. Cada DoritoImage se encarga, con sus propios métodos, de tratar el paquete RTP, extraer la información y avisar cuando la imagen está completa.

Hay que tener en cuenta que, las imágenes recibidas son el resultado de la unión de la pareja de imágenes, por la fuente. Por lo que en recepción se deben tratar como si de una sola imagen se tratase. Los contenedores DoritoImage siguen un protocolo de actuación muy concreto para administrar los paquetes recibidos. Para cada paquete sigue los siguientes pasos:

1. Extrae la cabecera RTP y la cabecera específica del RFC 4175.

2. Extrae los datos de la imagen. Si es el primer paquete, configura el DoritoImage con los datos del payload y otros de la aplicación.

3. Comprueba el timestamp de la cabecera:

    a. Si es menor al de la imagen actual, descarta el paquete. Se da por hecho que es un paquete de una imagen ya mostrada, que ha llegado tarde.

    b. Si es mayor al de la imagen actual, da por completada la imagen actual. Se da por hecho que se ha perdido el último paquete de la imagen actual y no se ha detectado su fin.

    c. Si es el mismo de la imagen actual, sigue al punto 4.

4. Añade al buffer de la imagen los datos recibidos, teniendo en cuenta los campos de nº de línea, offset y tamaño para situarlos en su sitio.

5. Incrementa en uno los paquetes utilizados para formar la imagen actual.

6. Comprueba si el campo Marker de la cabecera tiene valor "1". De ser así, este paquete corresponde al último de la imagen. Se da por completada la imagen actual.

Los contenedores DoritoImage contienen los siguientes parámetros:

- buffer para almacenar la imagen YUV 4:2:2 - UYVY
- timestamp
- resolución
- modo de imagen (Side-by-Side o Side-by-Side extendido)
- número de paquetes integrados en la imagen

2.3.3 Reproducción

El apartado de reproducción es, a la práctica, el que más aprovecha el código heredado de la aplicación v4l2stereo. Se ha aplicado el sistema de reproducción de imágenes mediante OpenCV. En esta ocasión, la transformación a IplImage la realiza un método del contenedor DoritoImage.

El tamaño de la ventana se configura con los datos del payload, extraídos del primer paquete recibido. Al recibir el par de imágenes como si fuesen una sola, se muestran juntas en la misma ventana. En la figura 2.19 aparecen dos capturas de la aplicación, cada una recibiendo el vídeo de un modo, y la fuente original. Se puede comprobar cómo cada ventana se adapta al formato.

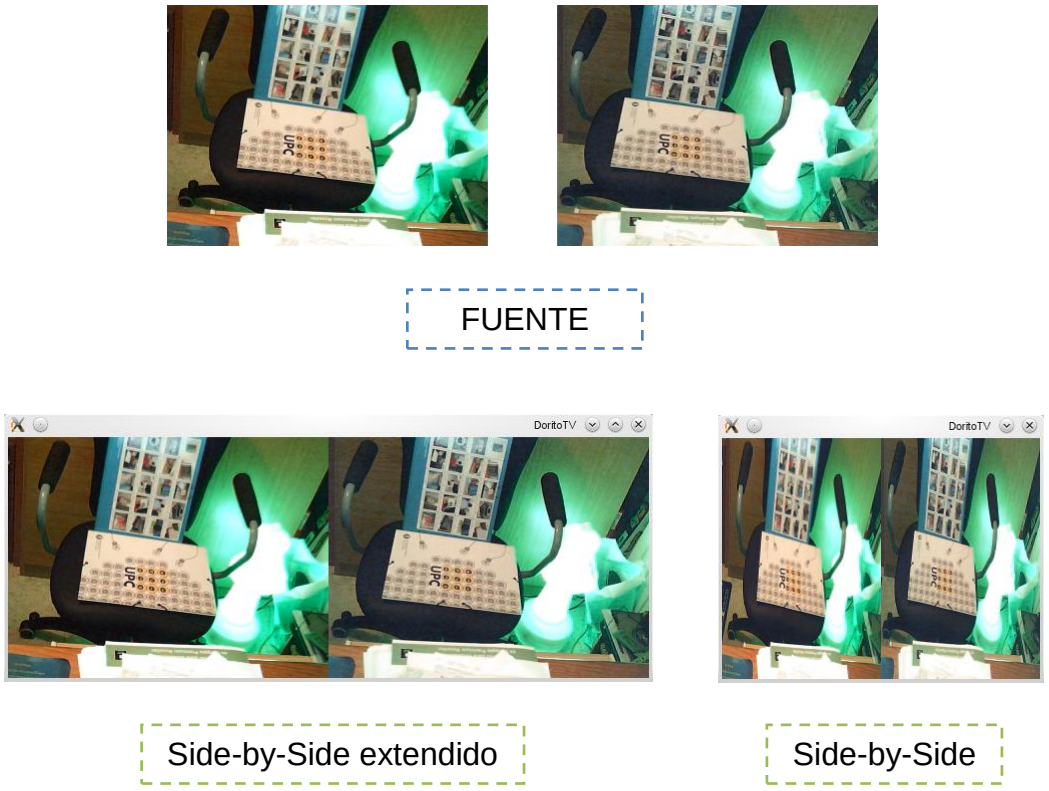


Figura 2.19 DoritoTV reproduciendo fuente con diferente modo

Para configurar ciertos aspectos de la reproducción, se han incluido un par de flags al arrancar la aplicación. Son los siguientes:

- "`--reescalar`": Activa la opción de reescalado. Esta opción provoca que, si el vídeo se recibe en modo Side-by-Side, se muestre el vídeo reescalado al tamaño original. El efecto es como si se recibiese en modo Side-by-Side extendido, pero con peor calidad de imagen.

- "`--nowindow`": Desactiva la visualización del vídeo en pantalla, pero no detiene la recepción. Esta función existe para comprobar hasta qué punto la carga del sistema afecta a la pérdida de paquetes.

El reescalado se realiza en el proceso de conversión de la imagen a RGB, por el DoritoImage, doblando el número de columnas. En la figura 2.20 se muestra el proceso en un esquema. No ofrece un gran resultado, ya que da la impresión de que toda la imagen está pixelada, pero permite ahorrar la mitad del ancho de banda.

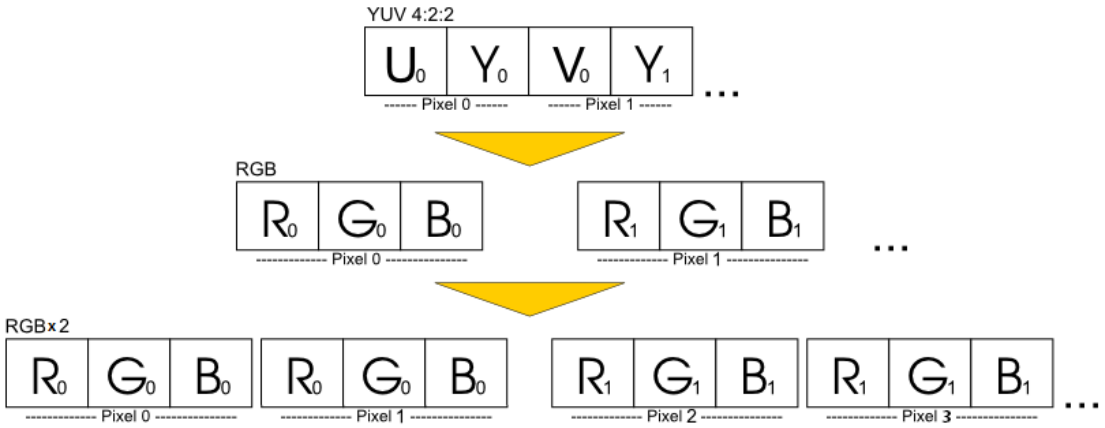


Figura 2.20 Reescalado UYVY -> RGBx2

CAPÍTULO 3. EVOLUCIÓN DEL DISEÑO Y PRUEBAS

Este capítulo sigue la evolución del diseño, y las pruebas realizadas, durante la implementación y mejora de todos los elementos del proyecto. Se han realizado con equipos modestos, no excepcionalmente potentes, por lo que se ha tenido que afinar mucho al implementar cada función. Al conseguir un buen rendimiento en estos equipos, se asegura que el sistema no requiere un hardware de precio prohibitivo para funcionar, y que se pueden añadir mejoras teniendo margen en cuanto a necesidades de proceso.

Los equipos utilizados tienen las características de la tabla 3.1:

Procesador	Intel Pentium D, 64 bits, ~5 años Intel Core 2 Duo, 64 bits, ~4 años
Memoria RAM	2 GB
Sistema operativo	Open Suse 11.3, 32 o 64 bits

Tabla 3.1 Características de los equipos

Los principales problemas que se han encontrado han sido el tiempo de proceso y el ancho de banda. Al tratarse de vídeo en tiempo real sin comprimir, cada pareja de imágenes tiene, para procesarse y enviarse, el periodo que tarda la cámara en entregarle la siguiente pareja de imágenes y, a la vez, un tamaño excesivo para una conexión fuera de red local. En la tabla 3.2 se pueden ver los cálculos, para darse cuenta de hasta qué punto el sistema tiene que ir ligero y no perder el tiempo en acciones segundarias. En el anexo B aparecen los cálculos de ancho de banda para todas las opciones.

Cámara	Resolución	FPS	Periodo (s)	TpB (s)	Vtx (Mbit/s)
Minoru 3D	320 x 240	30	0.033	10.85^{-8}	76
	640 x 480	15	0.066	5.42^{-8}	153
Videre	640 x 480	25	0.040	2.17^{-8}	255
	1280 x 960	7	0.143	1.93^{-8}	285

Tabla 3.2 Datos teóricos de proceso y flujo en modo Side-by-Side extendido

Explicación de varios campos de la tabla:

- TpB (s): relaciona cuantos segundos tiene la aplicación, para procesar y enviar cada Byte obtenido por las cámaras.

- Vtx (Mbit/s): velocidad de transferencia en Mbits por segundo, teniendo en cuenta las cabeceras.

Los problemas de ancho de banda, al no poder comprimir el flujo, solo se pueden solucionar utilizando conexiones con suficiente ancho de banda. En las pruebas se han utilizado switchs Gigabit Ethernet.

3.1 Diseño inicial

El diseño inicial del sistema era completamente secuencial. Tanto el bloque de adquisición, como el bloque de reproducción, iban paso a paso. No había ningún control de proceso. En la figura 3.1 se muestran ambos diseños.

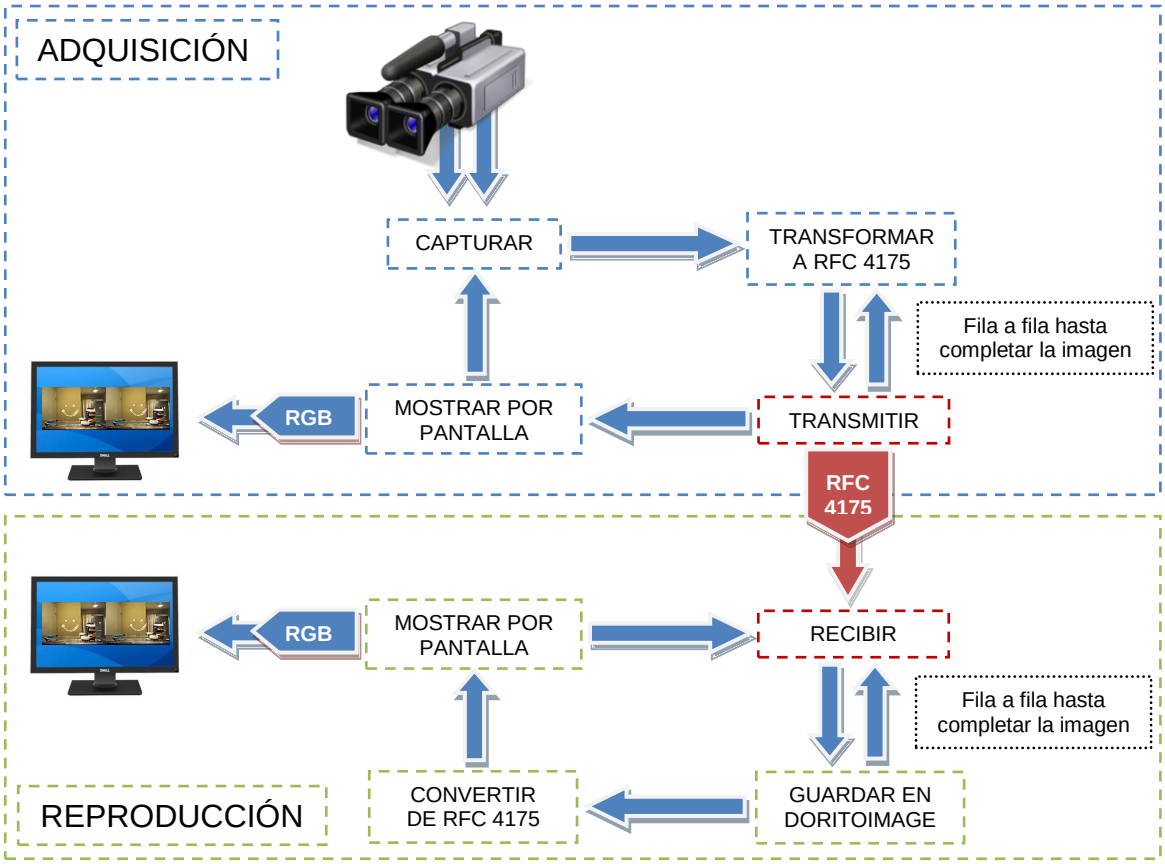


Figura 3.1 Diseño inicial

Se puede observar como el bloque de adquisición primero captura la pareja de imágenes, la transforma según el RFC 4175 y la transmite por red fila a fila y, para finalizar, la muestra por pantalla. Por su parte, el bloque de reproducción recibe y guarda cada fila en un contenedor DoritoImage, hasta tener la pareja de imágenes completa, la convierte del RFC 4175 a RGB y la muestra por pantalla.

Las pruebas realizadas, con un Intel Core 2 Duo con Open Suse 11.3 de 32 bits, siguiendo este diseño con la Minoru 3D mostraron hasta un 93% de pérdidas de paquetes. El procesador no estaba saturado, por lo que no se podía achacar a falta de recursos. Vistos los resultados, no se realizaron pruebas con la Videre. Las pruebas de los siguientes apartados 3.2, 3.3, 3.4 y 3.5 se realizaron con la misma máquina.

La tabla con los datos de las pruebas aparece en el anexo A.

3.2 Recepción en grupos

Analizando los problemas del diseño inicial, se llegó a la conclusión de que, el sistema perdía paquetes porque no tenía tiempo de tratarlos. Los paquetes llegaban a la tarjeta de red del ordenador, el monitor de red lo confirmaba mostrando el flujo de entrada pertinente, pero el buffer del socket se saturaba y no entraban en la cola. El proceso de "recibir paquete -> guardar fila del paquete -> esperar otro paquete..." parecía muy lento.

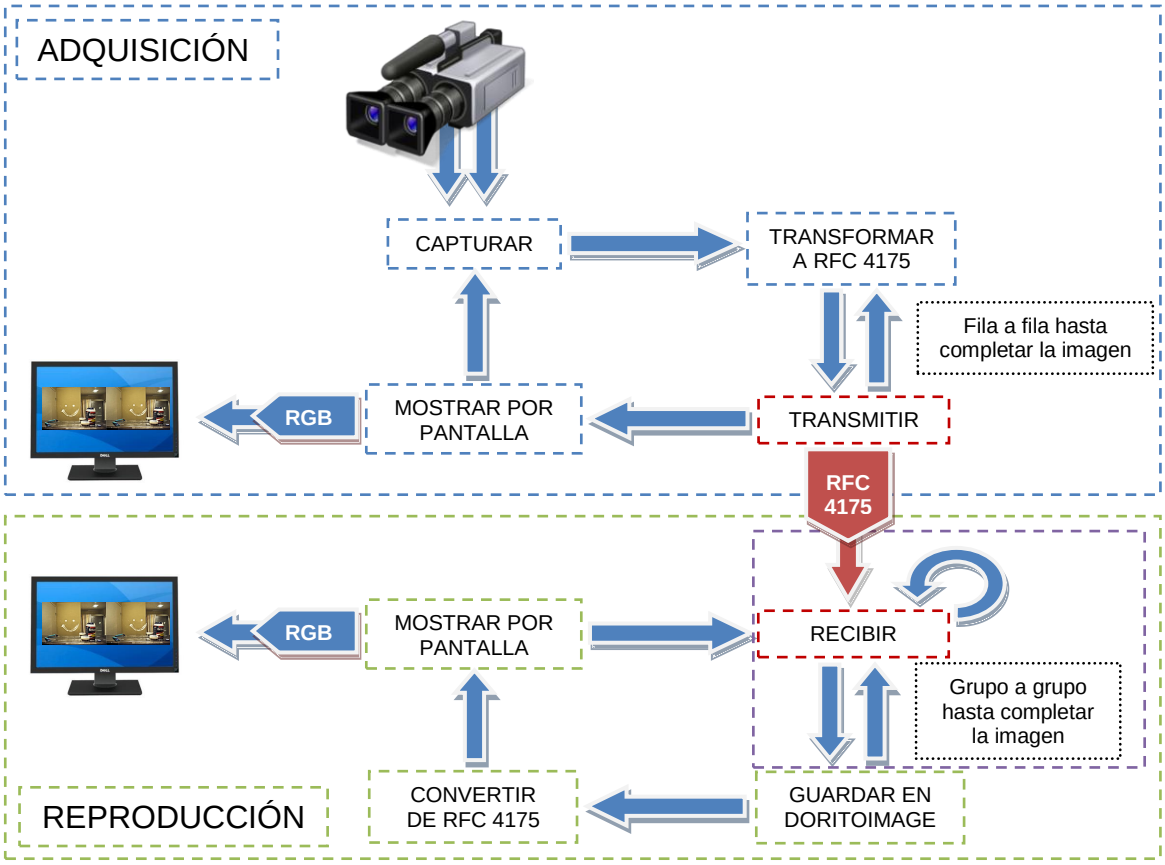


Figura 3.2 Diseño con grupos de paquetes en recepción

Para solucionar el problema se varió el diseño, en el proceso de recepción, tal y como aparece en la figura 3.2. Con esta modificación, el bloque de reproducción recibe paquetes y los almacena mientras detecte alguno en la cola. Cuando no hay paquetes a la espera, se pasan a almacenar en el contenedor DoritoImage.

Las pruebas realizadas muestran una mejora inapreciable a la vista, pero una subida importante en el consumo del sistema. Con este diseño se llegan a perder hasta un 90% de paquetes, pero se consume entre un 10 y un 20% más de CPU.

La tabla con los datos de las pruebas aparece en el anexo A.

3.3 Envíos espaciados

Viendo que las recepciones en grupo no arreglaron el problema, se decidió eliminar esa mejora y observar el bloque de adquisición. Al ser un sistema secuencial, cada vez que hay una imagen nueva, se pasa a transformar y retransmitir sin ninguna espera. Analizando el flujo de red, se observó que los paquetes se enviaban todos de golpe. Puesto que la cámara deja tiempo de sobra para enviar los paquetes, antes de crear una nueva pareja de imágenes, no es necesario enviar todos los paquetes lo más rápido posible, por lo que se pasó a espaciar los envíos.

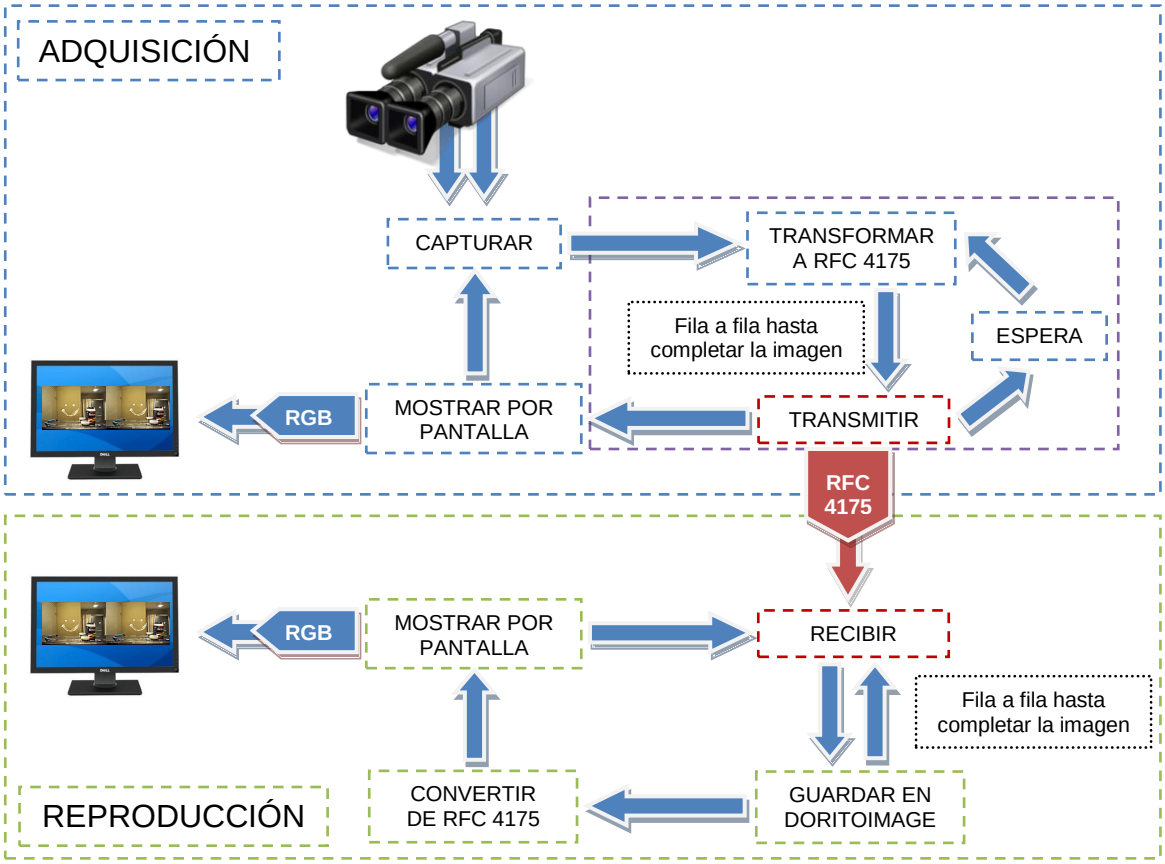


Figura 3.3 Diseño con envíos espaciados

En la figura 3.3 se observa, en el bloque de adquisición, cómo se realiza una espera entre transferencia y transferencia. De esta manera, los paquetes no llegan tan juntos, y se da más tiempo al bloque de reproducción a procesarlos antes de que le llegue otro.

Las pruebas realizadas muestran como el buffer del socket no se satura tanto. Los paquetes perdidos pasan a ser, en el peor de los casos, un 19%. Una gran mejora comparado con el 90% anterior, pero todavía insuficiente para un reproductor de vídeo, desde el punto de vista del usuario.

La tabla con los datos de las pruebas aparece en el anexo A.

3.4 Quad-Socket

Con las mejoras conseguidas al incluir los envíos separados, quedó claro que había que darle más tiempo al buffer del socket, para no saturarlo. Siendo más complejo mejorar la velocidad, se optó por ampliar la cantidad de buffers. Si un socket se saturaba, quizás 4 en paralelo con una cuarta parte de carga cada uno no se saturarían, por lo que se añadieron 3 sockets y las transferencias pasaron a realizarse por cuatro sockets distintos (en un puerto UDP diferente cada uno), tal como se aprecia en la figura 3.4.

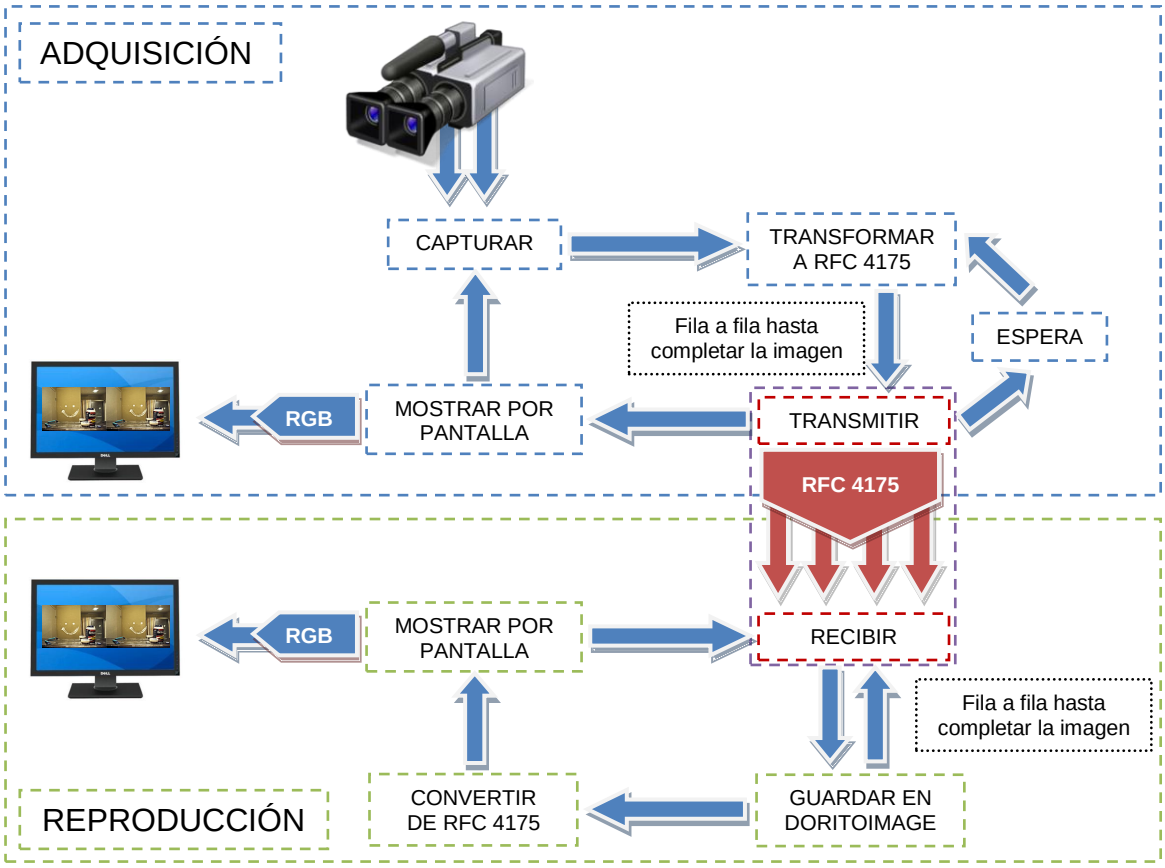


Figura 3.4 Diseño con el sistema Quad-Socket

En el bloque de adquisición, los sockets funcionan de manera secuencial. Cada fila se transmite por uno distinto, siguiendo la norma "número de fila / 4" para decidir por cual. El sistema de esperas dio buenos resultados, por lo que sigue apareciendo en este bloque. En el bloque de reproducción, los sockets se van turnando hasta que uno recibe un paquete, en ese momento el paquete se pasa a procesar. El sistema de recepción por grupos no dio buenos resultados, por lo que no está integrado.

Las pruebas realizadas muestran unas mejoras sustanciales. Con la Minoru 3D se pierden, en el peor de los casos, un 4%, y con la Videre un 2.6%. Todo y estas mejoras, no es suficiente, pero muestran el camino a seguir.

La tabla con los datos de las pruebas aparece en el anexo A.

3.5 Reproducción en paralelo

Con el diseño del Quad-Socket y las esperas en transmisión se consiguió un buen rendimiento, pero aun no era suficiente. El camino quedó claro, conseguir mayor rendimiento y que los sockets no se saturasen. Para conseguir que un socket no se sature, la mejor opción es dejar un proceso pendiente de él. El diseño que solucionó eso, y que pasó a ser el definitivo, fue el de la figura 2.5.

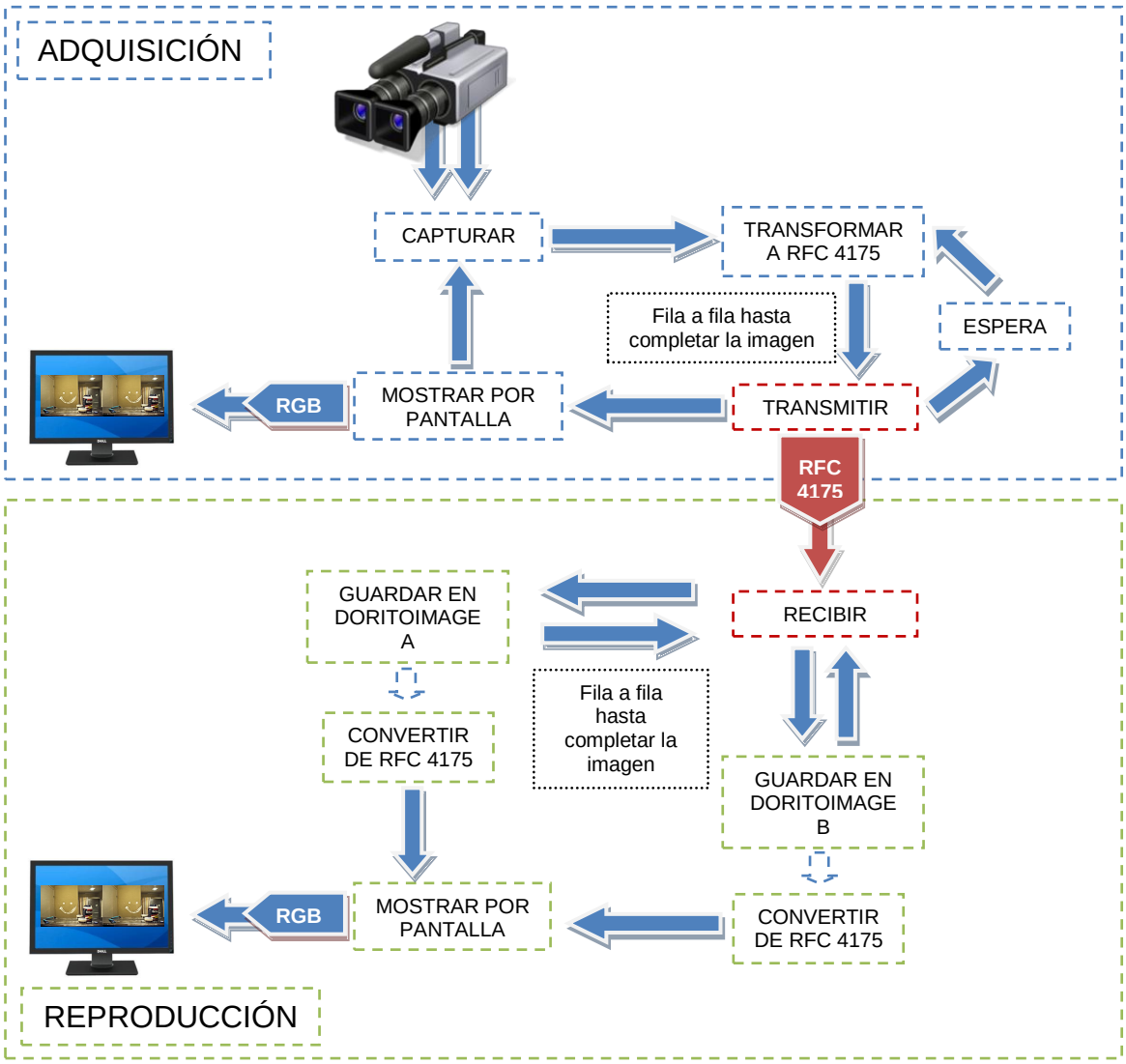


Figura 3.5 Diseño con reproducción en paralelo

El diseño final deja el bloque de adquisición como en el diseño con envíos espaciados. Esta solución da buenos resultados, y está implementada de modo que varía el tiempo de espera en función de las posibilidades. En cambio, el bloque de reproducción queda modificado completamente.

El bloque de reproducción pasa a ser multi-hilo. Se crea un hilo de proceso que se dedica a recibir los paquetes y guardarlos en un DoritoImage. Cuando el DoritoImage está preparado, cambia a un segundo DoritoImage para seguir almacenando los paquetes de la siguiente imagen. En paralelo, el hilo principal

de la aplicación detecta que el primer DoritoImage está preparado, realiza la conversión de la imagen de este y la muestra por pantalla. De esta manera se consigue que el socket siempre esté atendido y no llegue a saturarse.

Las pruebas realizadas dan los resultados esperados. Utilizando la Videre como fuente, las pérdidas son del 0% en todos los casos. La Minoru 3D ofrece menos problemas, por lo que tampoco tiene pérdidas. En el anexo A aparecen los datos de las pruebas realizadas con la Videre.

Es importante recordar que todas las pruebas se han realizado en ordenadores muy modestos sobre Gigabit Ethernet. Incluso algunas ejecutando todos los elementos en la misma máquina, como es el caso de los valores mostrados en estas pruebas, por lo que los recursos aun se limitan más.

3.6 Análisis de una transmisión

Para analizar las transmisiones y hacer pruebas se ha utilizado Wireshark [R9]. Hoy en día no existe ningún módulo para filtrar el RFC 4175, aunque hay una estudiante de la EETAC que lo está realizando [R10], pero sí de RTP. Por lo que se puede capturar un flujo y observar si los datos de configuración de cada paquete son los correctos. Esto ha sido muy útil a la hora de crear el reproductor, ya que primero hay que asegurarse que le llega todo bien.

A continuación se analiza una transmisión desde la Minoru 3D, a 640 x 480, en Side-by-Side extendido, sobre Gigabit Ethernet con MTU máxima de 1500 bytes y 960 paquetes por imagen con una línea por cada dos paquetes. En el anexo B se explica la distribución de las imágenes en paquetes.

```
Frame 1921 (1342 bytes on wire, 1342 bytes captured)
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
User Datagram Protocol, Src Port: 46884 (46884), Dst Port: 25225 (25225)
Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count: 0
    0... .... = Marker: False
    Payload type: DynamicRTP-Type-114 (114)
    Sequence number: 19924
    Timestamp: 3770079614
    Synchronization Source identifier: 0x885f6504 (2287953156)
    Payload: 03FA050000000000779867967998 6A9A79956A957A936A92...
```

El frame 1921 indica en su Payload el del tipo 114, que coincide con el declarado para el modo 640x480, en Side-by-Side extendido, en la tabla 2.1, que aparece al inicio del capítulo 2 de este documento. El resto de campos estáticos también coinciden, por lo que en el resto de capturas se ocultarán.

Se observa en el Payload la cadena "03FA..", estos son los bits de mayor peso del timestamp. Continúa con "..0500..", que indica el tamaño de bytes de la imagen contenida, que son 1280, y sigue con "..0000-0000..", que indican que es la fila cero y que no tiene offset, por lo que es el primer paquete de ésta.

```
Frame 1922 (1342 bytes on wire, 1342 bytes captured)
Real-Time Transport Protocol
    0... .... = Marker: False
    Sequence number: 19925
    Timestamp: 3770079614
    Payload: 03FA050000000280799A6A997B956C967C986D987D986C98...
```

El frame 1922 es el siguiente, y se puede observar como tiene el mismo timestamp, pero el siguiente número de secuencia. Esto es porque, tal como se explica en el apartado 1.2.2, el número de secuencia se incrementa en cada paquete. También se puede ver que sigue teniendo el mismo tamaño y sigue correspondiendo a la fila cero. En cambio, los campos "..0280.." indican que, como es de esperar, tiene un offset de 640 píxeles.

```
Frame 2879 (1342 bytes on wire, 1342 bytes captured)
Real-Time Transport Protocol
    0... .... = Marker: False
    Sequence number: 20882
    Timestamp: 3770079614
    Payload: 03FA050001DF000078C890C575C790D172D791DB70DD93DE...
```

Saltando hasta el frame 2879, este corresponde al primer paquete, de dos, de la última fila. Esto se demuestra porque indica "..01DF-000..", que corresponde a fila 479 y offset 0. También se puede comprobar que no se ha perdido ni un paquete, ya que restando los número de secuencia del frame 2879 y el 1922 y sus números de frame se consigue el mismo número, 957. Teniendo en cuenta que, para este modo de transmisión, hay 960 paquetes por imagen (cada uno de ellos transportando 1280 bytes de imagen, correspondientes a media línea), todo cuadra.

```
Frame 2880 (1342 bytes on wire, 1342 bytes captured)
Real-Time Transport Protocol
    1... .... = Marker: True
    Sequence number: 20883
    Timestamp: 3770079614
    Payload: 03FA050001DF028081A57C9083907495808F6FA67AA370B0...
```

El frame 2880 debe mantener el mismo timestamp que los anteriores, pero indicar que es el último de la imagen. Esto queda reflejado al estar activo su Marker. También se muestra que es la misma fila que la anterior "..01DF..", pero con offset, "..0280..", de 640 píxeles.

```
Frame 2881 (1342 bytes on wire, 1342 bytes captured)
Real-Time Transport Protocol
    0... .... = Marker: False
    Sequence number: 20884
    Timestamp: 3770104453
    Payload: 03FA0500000000007D956B987F976B957F946A947D946B94...
```

Para el último frame analizado, el 2881, se espera que sea el primero de una imagen nueva, por lo que el timestamp debe ser diferente, como vemos que ocurre. También indica que es la fila cero con offset cero "..0000-0000..", y se inicia de nuevo el ciclo de paquetes en que se fragmenta la segunda imagen de la sesión.

CAPÍTULO 4. CONCLUSIONES Y LÍNEAS FUTURAS

Frente a la creciente popularidad del vídeo en 3D, tanto en salas de cine como en el hogar, este TFC se ha centrado en realizar un escenario de adquisición, conversión, transmisión, distribución, recepción, reconversión y reproducción de vídeo 3D sin comprimir en tiempo real. Todo ello aprovechando aplicaciones de código abierto existentes, escritas en C/C++ para GNU/Linux.

Un escenario tan amplio y variado ha requerido tener que abordar muchos temas diferentes. Teniendo en cuenta que las transmisiones de vídeo en 3D están aún poco definidas y que, salvo casos como UltraGrid [anexo D] destinados a investigación, las transmisiones de vídeo sin comprimir no se contemplan, se ha tenido que investigar, idear y adaptar a nuestras necesidades aplicaciones y sistemas que no estaban pensados para el fin que se les ha dado.

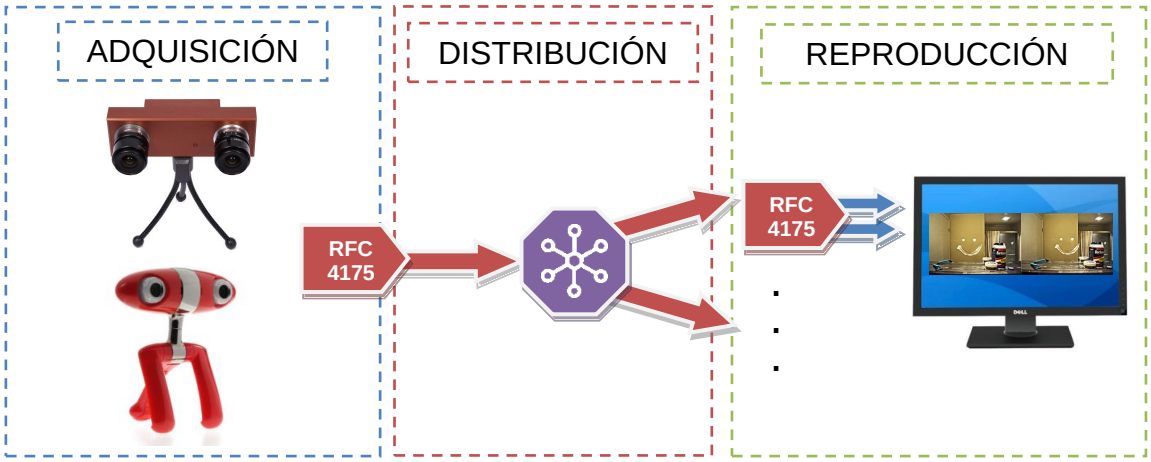


Figura 4.1 Escenario realizado

Para la adquisición se han utilizado dos cámaras estereoscópicas muy diferentes. Primero se ha tenido que aprender cómo funcionaban, ya que aplicaciones que realicen recepción y transmisión de vídeo estereoscópico sin comprimir no existen. Con la lección aprendida, se ha tenido que profundizar en el estándar RFC 4175 sobre RTP, para vídeo sin comprimir, para poder realizar las transferencias sobre su modelo.

Con las transmisiones fluyendo sobre un modelo estándar, y con ello asegurada su compatibilidad con otros sistemas, se ha tenido que investigar el mejor modo de recepción. Siendo un flujo tan pesado, el sistema de recepción empezó teniendo muchas pérdidas, hasta dar con la solución.

Teniendo un sistema adquisición/transmisión y recepción/reproducción listo, sólo faltaba un replicador, para poder enviar el flujo a grupos de clientes que lo deseasen. Por ello se creó un replicador de paquetes, con un sistema de mensajes compatible con los sistemas de recepción/reproducción, para poderse conectar desde un receptor/reproductor a modo de cliente.

El sistema de señalización no fue la primera opción ni la ideal. Se intentó adaptar un sistema SIP para gestionar los módulos (ver anexo F), pero al estar mayoritariamente dedicado a sistemas de telefonía IP y no encontrar ningún modo sencillo de adaptarlo al sistema, se optó por la creación de mensajes sencillos comentada.

Se han realizado pruebas de rendimiento, tanto analizando paquetes mediante Wireshark, como mostrando los paquetes que llegaban a la aplicación, para detectar si el escenario funcionaba correctamente y, en caso de no ser así, qué podía estar pasando. Estas pruebas han sido el centro de información a la hora de realizar las mejoras, hasta conseguir un rendimiento óptimo.

Finalmente, se ha obtenido un escenario de adquisición, distribución y reproducción de vídeo 3D sin comprimir preparado para el siguiente paso. La evolución lógica se está realizando ya, y son dos TFC de dos estudiantes. Uno centra sus esfuerzos en realizar un reproductor 3D con verdadero efecto 3D, aprovechando el kit 3D de "nVidia" [R7]. El otro estudiante está investigando como comprimir el flujo de vídeo, observando tecnologías que van apareciendo, sin en tiempo real [R6]. Otra mejora sería adaptar la señalización al estándar SIP, con el reto de desarrollar una extensión del protocolo SDP (Session Description Protocol) para señalizar las sesiones que incluyan flujos estereoscópicos.

Desde el punto de vista medioambiental, se pueden hacer algunas consideraciones. El hecho de transmitir dos vistas, y sin comprimir, consigue que el consumo de red se dispare, y con ello el consumo de los elementos que incluyen la red. Por otro lado, el acercar una videoconferencia 3D a cualquier punto de una red local puede ahorrar desplazamientos, aunque será más interesante cuando pueda salir a la red pública y alcanzar su máximo despliegue global. Las aplicaciones multimedia son las que más consumen, después de los videojuegos, por ello los sistemas se tienen que optimizar al máximo.

Durante la realización del proyecto se ha interactuado con i2Cat para poner conceptos en común. i2Cat, entre otras cosas, está desarrollando el UltraGrid y se les cedió el módulo que controla la cámara Videre para que lo integrasen en el. A la vez, se pudo observar cómo funcionaba UltraGrid ya que, a fin de cuentas, nuestro escenario se puede entender como un "mini UltraGrid". Estas relaciones se comentan en el anexo D.

BIBLIOGRAFÍA

Cámaras

[C1] Web oficial de la Minoru 3D Webcam - http://www.minoru3d.com
[C2] Web oficial del driver V4L2 - http://v4l2spec.bytesex.org
[C3] Web del proyecto Sentience - http://code.google.com/p/sentience
[C4] Web del proyecto libv4l2cam - http://code.google.com/p/libv4l2cam
[C5] Manual para la aplicación v4l2stereo - http://code.google.com/p/sentience/wiki/MinoruWebcam
[C6] Web oficial de Videre Design - http://www.videredesign.com

Red

[N1] RTP, RFC 3550 - http://tools.ietf.org/html/rfc3550
[N2] RTP, Wikipedia - http://en.wikipedia.org/wiki/Real-time_Transport_Protocol
[N3] Librería rtp.c - http://www.asterisk.org/doxygen/asterisk1.0/rtp_8c.html
[N4] RFC 4175 - http://tools.ietf.org/html/rfc4175
[N5] Web de Julian Highfield - http://spirit.lboro.ac.uk/
[N6] UDP Packet Reflector Hacks - http://spirit.lboro.ac.uk/mug/mug.html

Imagen

[I1] YUV, fourcc - http://www.fourcc.org/yuv.php
[I2] YUV <-> RGB, fourcc - http://www.fourcc.org/fccyvrgb.php
[I3] YUV, Wikipedia - http://en.wikipedia.org/wiki/YUV
[I4] RGB, Wikipedia - http://en.wikipedia.org/wiki/RGB_color_model
[I5] RGBA, Wikipedia - http://en.wikipedia.org/wiki/RGBA_color_space
[I6] Pruebas 3D TV3, tv3.cat - http://www.tv3.cat/3d
[I7] Estándar Side-by-Side para la 3DTV, DVB Document A154 Anexo A - http://www.dvb.org/technology/standards/a154_DVB-3DTV_Spec.pdf

Vista

[V1] Visión binocular, Wikipedia - http://es.wikipedia.org/wiki/Visión_binocular
[V2] Estereoscopía, Wikipedia - http://en.wikipedia.org/wiki/Stereoscopy
[V3] Sensación y Percepción, Eddie Marrero - http://academic.uprm.edu/~eddiem/psic3001/id61.htm
[V4] Handheld Viewers - http://www.ignomini.com/photographica/3dhandviewers.html
[V5] Cómo funciona la tecnología 3D - http://www.ennegrita.com/2010/02/17/como-funciona-la-tecnologia-3d-explicacion/
[V6] Better glasses-free 3-D, MIT news - http://web.mit.edu/newsoffice/2011/glasses-free-3d-0504.html

[V7] Parallax barrier, Wikipedia -
http://en.wikipedia.org/wiki/Parallax_barrier
[V8] Pantallas auto-estereoscópicas, Wikipedia -
http://en.wikipedia.org/wiki/Autostereoscopic_display

SIP

[S1] SIP, Wikipedia -
http://en.wikipedia.org/wiki/Session_Initiation_Protocol
[S2] oSIP - http://www.gnu.org/software/osip
[S3] eXosip - http://savannah.nongnu.org/projects/exosip
[S4] Live555 - http://www.live555.com/liveMedia
[S5] Javax SIP -
http://www.java2s.com/Open-Source/Java-Document/6.0-JDK-Modules/Java-Advanced-Imaging/javax.sip.htm
[S6] Cliente SIP i2Cat -
http://wiki.i2cat.net/doku.php/i2cat:public:clusters:audiovisual:uhdgroup:sipclient

Otras referencias

[R1] Ultra HD, Wikipedia -
http://es.wikipedia.org/wiki/Ultra_High_Definition_Television
[R2] UHDTV, HDTV Magazine -
http://www.hdtvmagazine.com/columns/2011/04/hdtv-almanac-smpte-develops-new-hdtv-format.php
[R3] Proyecto Vision, Panorama audiovisual.com -
http://www.panoramaaudiovisual.com/es/2011/02/28/proyecto-vision-telepresencia-inmersiva-con-comunicaciones-de-vídeo-de-nueva-generacion/
[R4] Ruben Fernández Gutiérrez y Antonio Jesús Román Rodríguez, "Adquisición y visualización de vídeo 3D", Noviembre 2010, TFC EETAC UPC - http://upcommons.upc.edu/pfc/handle/2099.1/10651
[R5] Sergio González Fernández, "Compresión y transporte de televisión 3D sobre redes IP", Octubre 2010, TFC EETAC UPC -
http://upcommons.upc.edu/pfc/handle/2099.1/9973
[R6] Aleix Paradell Navarro, "Optimització d'un codificador de vídeo 3D" (título provisional), TFC EETAC UPC, en desarrollo
[R7] Andrés Lucas Enciso, "Desarrollo de un visualizador de vídeo 3D con gafas activas" (título provisional), TFC EETAC UPC, en desarrollo
[R8] Sergio López Romera, "3D sobre redes P2P" (título provisional), TFC EETAC UPC, en desarrollo
[R9] Analizador de tráfico de redes WireShark - http://www.wireshark.org
[R10] Lorena Laguarta, Guia de instalación e implementación de un disector para Wireshark, Prácticas en empresa EETAC UPC, Mayo 2010, pendiente de publicación en UPCommons
[R11] UltraGrid -
http://wiki.i2cat.net/doku.php/i2cat:public:clusters:audiovisual:uhdgroup:ultragrid

GLOSARIO

FPS	Frames Per Second
GNU	GNU is Not Unix
GPL	General Public License
HD	High Definition
HDTV	High Definition Television
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LCD	Liquid Crystal Display
MIT	Massachusetts Institute of Technology
MVC	Multiview Video Coding
RFC	Request For Comments
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
SBS	Side-by-Side
SBS Ex	Side-by-Side Extendido
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UHDTV	Ultra High Definition Television
USB	Universal Serial Bus
V4L	Video for Linux
VoIP	Voz sobre IP

ANEXO A: TABLAS DE LAS PRUEBAS REALIZADAS

Los cálculos siguientes se han realizado sobre un Core 2 Duo 6600 con Open Suse 11.3 32 bits. Las medidas se han tomado con transmisiones de 10 segundos, una vez estabilizadas las estadísticas.

Pruebas con diseño inicial:

Cámara utilizada	Minoru 3D	Minoru 3D	Minoru 3D	Minoru 3D
FPS	30	30	15	15
Resolución	320x240	320x240	640x480	640x480
Modo	SBS Ex	SBS	SBS Ex	SBS
CPU ocupada en total	21%	20%	25%	30%
Se muestra el vídeo?	SI	SI	SI	SI
Paquetes por imagen esperados	240	240	960	480
Tamaño de un paquete en bytes	1280	640	1280	1280
Media de paquetes por imagen recibidos	71	119	71	71
% de paquetes perdidos	**70**	**50**	**93**	**85**

Pruebas con diseño con grupos de paquetes en recepción

Cámara utilizada	Minoru 3D	Minoru 3D	Minoru 3D	Minoru 3D
FPS	30	30	15	15
Resolución	320x240	320x240	640x480	640x480
Modo	SBS Ex	SBS	SBS Ex	SBS
CPU ocupada en total	32%	27%	50%	40%
Se muestra el vídeo?	SI	SI	SI	SI
Paquetes por imagen esperados	240	240	960	480
Tamaño de un paquete en bytes	1280	640	1280	1280
Media de paquetes por imagen recibidos	80,36	129,27	95,36	86,65
% de paquetes perdidos	**67**	**46**	**90**	**82**

Pruebas con diseño con envíos espaciados

Cámara utilizada	Minoru 3D	Minoru 3D	Minoru 3D	Minoru 3D
FPS	30	30	15	15
Resolución	320x240	320x240	640x480	640x480
Modo	SBS Ex	SBS	SBS Ex	SBS
CPU ocupada en total	35%	30%	50%	38%
Se muestra el vídeo?	SI	SI	SI	SI
Paquetes por imagen esperados	240	240	960	480
Tamaño de un paquete en bytes	1280	640	1280	1280
Media de paquetes por imagen recibidos	238,00	240	777,43	480
% de paquetes perdidos	**0,83**	**0**	**19**	**0**

Pruebas con diseño con sistema Quad-Socket

Cámara utilizada	Minoru 3D	Minoru 3D	Minoru 3D	Videre	Videre
FPS	30	15	15	25	6
Resolución	320x240	640x480	640x480	640x480	1280x960
Modo	SBS Ex	SBS Ex	SBS Ex	SBS Ex	SBS Ex
CPU ocupada en total	38%	54%	54%	66%	61%
Se muestra el vídeo?	NO	NO	SI	SI	SI
Paquetes por imagen esperados	240	960	960	960	3840
Tamaño de un paquete en bytes	1280	1280	1280	1280	1280
Media de paquetes por imagen recibidos	240	920,74	960	955	3738,59
% de paquetes perdidos	**0**	**4,09**	**0**	**0,52**	**2,64**

Pruebas con el diseño con reproducción en paralelo

Cámara utilizada	Videre	Videre
FPS	25	6
Resolución	640x480	1280x960
Modo	SBS Ex	SBS Ex
CPU ocupada en total	64%	66%
Se muestra el vídeo?	SI	SI
Paquetes por imagen esperados	960	3840
Tamaño de un paquete en bytes	1280	1280
Media de paquetes por imagen recibidos	960	3840
% de paquetes perdidos	**0**	**0**

ANEXO B: CÁLCULOS DE ANCHO DE BANDA

Estas tablas muestran los valores teóricos de ancho de banda a nivel IP y la distribución de las imágenes en paquetes. Se tiene en cuenta una MTU máxima de 1500 bytes, que es la típica para Ethernet.

Resolución	Cámara	FPS	Modo	Vtx (Mbit/s)
320 x 240	Minoru 3D	30	SBS	**39'63**
			SBS Ex	**76'49**
	Videre	25	SBS	**33'02**
			SBS Ex	**63'74**
640 x 480	Minoru 3D	15	SBS	**76,49**
			SBS Ex	**152'98**
	Videre	25	SBS	**127'49**
			SBS Ex	**254'98**
1280 x 960	Videre	7	SBS	**142'79**
			SBS Ex	**285'57**

Cada imagen se divide en líneas, y cada línea se divide en bloques de 1280 bytes, que se envían individualmente en cada paquete. Existe la excepción del caso 320 x 240 en Side-by-Side, que cada línea son 640 bytes y se envían en paquetes individuales. Esto es así porque es el caso con peor calidad y no se planteó hacer agregación de filas para aprovechar el paquete.

Resolución	Modo	ppi *	bytes paquete	bytes útiles
320 x 240	SBS	240	688	640
	SBS EX	240	1328	1280
640 x 480	SBS	480	1328	1280
	SBS EX	960	1328	1280
1280 x 960	SBS	1920	1328	1280
	SBS EX	3840	1328	1280

*ppi = paquetes por imagen - fragmentos de una imagen completa.

Comprobación de cálculos realizados para conseguir el flujo en Side-by-Side extendido, a 320 x 240, con la Minoru 3D a máximo framerate, comparando con una captura real:

- Cálculo del tamaño del paquete a nivel IP

$$cabecera\ IP + cabecera\ UDP + cabecera\ RTP + cabecera\ RFC + bytes\ útiles = bytes\ paquete$$

$$20\ bytes + 8\ bytes + 12\ bytes\ + 8\ bytes + 1280\ bytes = \mathbf{1328\ bytes}$$

- Cálculo del número de paquetes por imagen

$$(alto * ancho * bytes\ por\ pixel * modo)/(Bytes\ útiles) = nº\ paquetes\ por\ imagen$$

$$(320 * 240 * 2 * 2)/1280 = \mathbf{240\ paquetes}$$

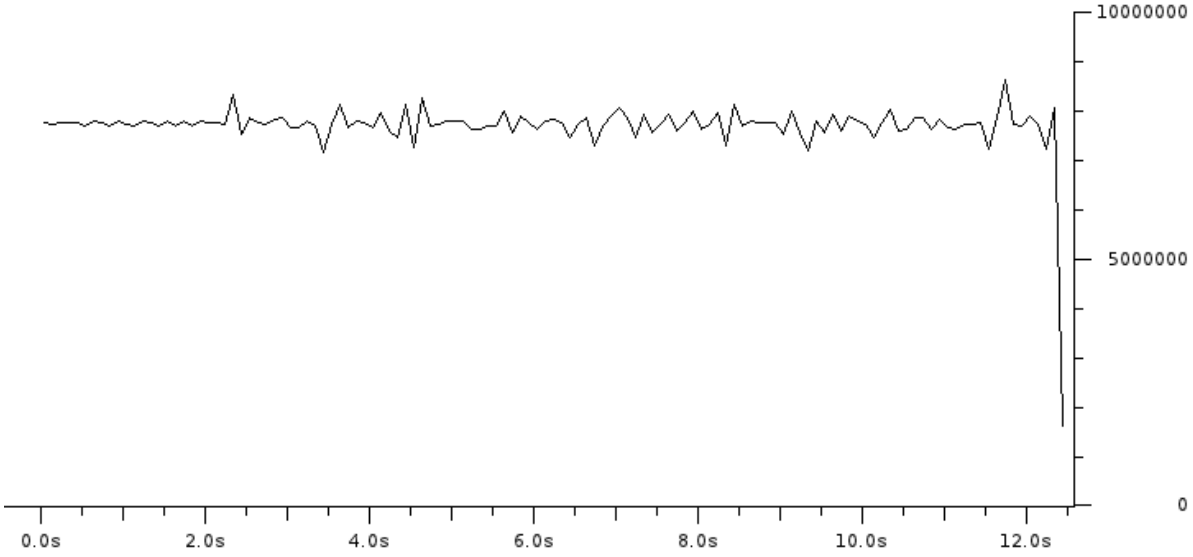
- Cálculo del ancho de banda necesario

$$nº\ paquetes * bytes\ paquete * fps * \frac{8\ bits}{1\ byte} * \frac{1\ Mbit}{1000\ bits} = ancho\ de\ banda\ en\ Mbit/s$$

$$(240 * 1328 * 30 * 8) / 1000000 = \mathbf{76'49\ Mbit/s}$$

- Comparando con captura de Wireshark

La siguiente gráfica pertenece a una captura de Wireshark (IO Graphs, ancho de banda medido en Mbit/s a nivel Ethernet) de las mismas condiciones que el ejercicio. Se puede observar que la media de la transmisión coincide con el ancho de banda calculado, aunque se queda un poco por debajo. Los cálculos nos dan 76'49 Mbit/s y la gráfica muestra unos 77-78 Mbit/s. Esto es porque los cálculos realizados son a nivel IP y Wireshark trabaja a nivel Ethernet.



Wireshark calcula 14 bytes de cabecera Ethernet, si se realiza el cálculo de ancho de banda sumándoselos, nos salen 77'29 Mbit/s.

$$(240 * (1328 + 14) * 30 * 8) / 1000000 = \mathbf{77'29\ Mbit/s}$$

ANEXO C: SEÑALIZACIÓN

La señalización se transmite con paquetes UDP enviados desde el cliente. Según el contenido el repetidor interpreta una u otra petición. Existen 3 mensajes distintos:

- "BYE" en ASCII (425945_{16}). El cliente indica que se desconecta y no desea recibir más flujo.

- "ACK" en ASCII ($41434B_{16}$). El cliente indica que desea seguir recibiendo más flujo. Este mensaje es necesario, ya que el repetidor elimina automáticamente, de su lista de clientes, a los que no lo envían en un periodo de más de 2 minutos. Es similar a un "keepalive".

- Otro contenido. Se utiliza para darse de alta en el repetidor. El contenido debe ser el puerto al que el repetidor enviará el flujo.

A continuación se explica paso a paso una captura real, como ejemplo de los paquetes que envían dos clientes al servidor, para pedir ser añadidos en la lista de clientes, para continuar en la lista y para ser eliminados.

- IP del repetidor: 147.83.118.253 (R)
- IP de los clientes: 147.83.118.135 (A) y 147.83.118.249 (B)

Nº	Time	Source	Destination	Protocol	Data
1	0.000000	147.83.118.135	147.83.118.253	UDP	6289
2	5.576256	147.83.118.249	147.83.118.253	UDP	6289
3	50.009254	147.83.118.135	147.83.118.253	UDP	425945
4	66.522608	147.83.118.249	147.83.118.253	UDP	41434B
5	70.779707	147.83.118.249	147.83.118.253	UDP	425945

Paquete 1: El cliente A envía el mensaje "6289_{16}" a R. Corresponde a la petición de recibir el flujo en su puerto 25225_{10}. R empieza a emitir hacia el puerto 25225 de A.

Paquete 2: El cliente B envía el mensaje "6289_{16}" a R. Corresponde a la petición de recibir el flujo en su puerto 25225_{10}. R empieza a emitir hacia el puerto 25225 de B.

Paquete 3: El cliente A envía el mensaje "425945_{16}" a R. Corresponde al "BYE". R lo interpreta como mensaje de fin de transmisión y elimina a A de su lista. R deja de transmitir hacia A.

Paquete 4: El cliente B envía el mensaje "$41434B_{16}$" a R. Corresponde al "ACK". R confirma la permanencia de B durante 2 minutos más.

Paquete 5: El cliente B envía el mensaje "425945_{16}" a R. Corresponde al "BYE". R lo interpreta como mensaje de fin de transmisión y elimina a B de su lista. R deja de transmitir hacia B.

ANEXO D: ULTRAGRID

UltraGrid [R11] es un sistema de distribución de vídeo en alta definición. Al igual que el sistema diseñado en este proyecto, permite realizar transmisiones de vídeo en tiempo real sin comprimir, pero de una manera mucho más profesional y con más posibilidades, aunque sin entrar en el apartado de reproducción. Entre otras cosas, UltraGrid convierte las señales de vídeo que recibe de las cámaras y las empaqueta en RTP siguiendo el RFC 4175.

UltraGrid fue desarrollado por el ISI East, pero ser de código libre ha permitido que la fundación i2Cat esté trabajando con él. Durante la realización de este proyecto fin de carrera hubo una serie de interacciones con i2Cat, para poner conceptos en común. Se tuvo la oportunidad de ver cómo funcionaba UltraGrid y se entregaron los códigos que controlaban, tanto la cámara Videre como la Minoru 3D, con todas las instrucciones necesarias para que los pudiesen integrar en su sistema.

Aprovechando el acercamiento para integrar código de este proyecto en UltraGrid, se pudo comprobar su funcionamiento y lo amplio que es. Tiene compatibilidad con multitud de cámaras, sistemas de entrada de vídeo y protocolos distintos. En i2Cat se han desarrollado extensiones que integran el protocolo de inicio de sesión SIP [S6].

En cierto modo, se podría referenciar este TFC como la creación de un "mini UltraGrid", sin tantos añadidos y más dedicado a unos casos en concreto. El problema de UltraGrid es que es muy complejo, y para necesidades básicas es excesivo, por lo que un sistema más sencillo pero optimizado para nuestro escenario concreto puede ser mucho más recomendable.

ANEXO E: CÁMARA STEREO VIDERE STH-MDCS2-C

E.1 Equipo utilizado

- Cámara stereo VIDERE Design
  - modelo STH-MDCS2-C
  - S/N 1618
  - conexión IEEE 1394 de 6 pines
  - resolución máxima 1280H x 960V px

- PC lab 325
  - Intel D 3GHz 64 bits
  - 2 GB RAM
  - nVidia GeForce 7300 GS
  - openSUSE 10.2 64bits

- Cable IEEE 1394 de 6 pines

E.2 Software necesario

SRI SVS Capture para Linux 2.4.x / 2.6.x

- Página web:
  `http://www.videredesign.com/index.php?id=121` (no es necesaria)

- Zona de descarga:
  `http://www.videredesign.com/assets/files/downloads/svslnx/`

SRI SVS Capture para Windows (en este documento no se tendrá en cuenta)

- Página web:
  `http://www.videredesign.com/index.php?id=122`

- Zona de descarga:
  `http://www.videredesign.com/assets/files/downloads/svsmsw`

E.3 Instalación

Descarga e instalación

1. Descargar el archivo svs44g.tgz del link indicado en el apartado *Software necesario.*

2. Descomprimirlo donde se quiera dentro de */home*, quedará dentro de la carpeta svs44g. A partir de aquí se da por hecho que el contenido del *tgz* está en /home/nombre_usuario/svs44g/

3. Descargar los drivers de la tarjeta gráfica desde la web del fabricante y ejecutarlos para que se instalen. Esto tiene que hacerse porque, en el caso probado, los drivers de nVidia que vienen con el sistema carecen de la librería libGL.so.1, y de paso se mejora el soporte gráfico del sistema.

4. Abrir *Yast2* y ejecutar la *Búsqueda de paquetes* (webpin).

5. Asegurarse de que no haya problemas con los repositorios.

6. Buscar e instalar *mesa-32bits*, *mesa-devel-32bits* y *libraw1394-8-32bits* con las dependencias que requieran para funcionar.

Preparar el medio de ejecución

1. Abrir la terminal desde la que se va a configurar y ejecutar la aplicación.

2. Incluir las librerías necesarias que se encuentran dentro del directorio en la variable LD_LIBRARY_PATH con el siguiente comando:

```
#> export LD_LIBRARY_PATH=/home/nombre_usuario/svs44g/bin:$LD_LIBRARY_PATH
```

3. Hay que tener en cuenta que el paso 2 es temporal, y solo afecta a la terminal en la que se ejecuta el comando. Si se cierra la terminal se tiene que volver a realizar en la terminal que se quiera usar para ejecutar la aplicación.

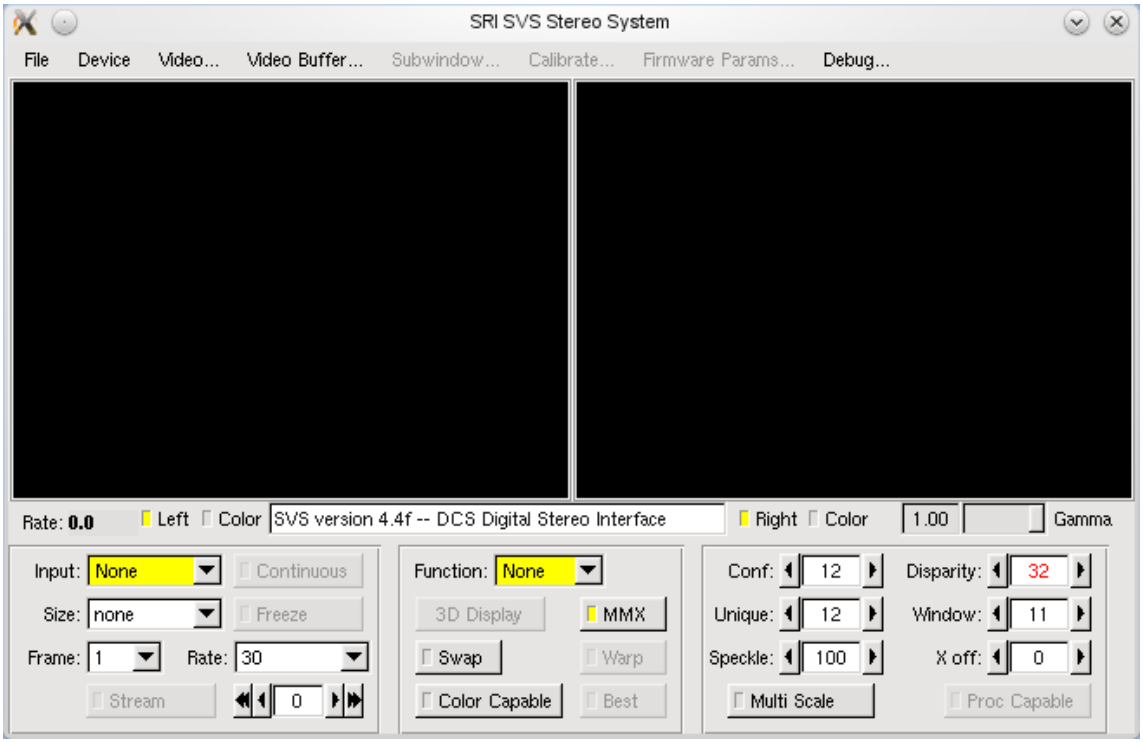
4. Antes de ejecutar la aplicación es recomendable conectar la cámara al ordenador, para que esta la pueda reconocer al arrancar.

E.4 Aplicación SVS Stereo System

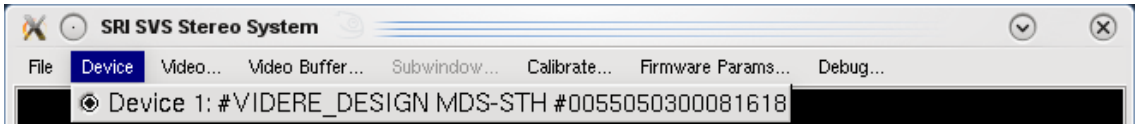
Para ejecutar la aplicación hay que utilizar la terminal configurada en el apartado anterior en “Preparar el medio de ejecución”. Se encuentra en la carpeta /home/nombre_usuario/svs44g/bin/ con el nombre *smallv*, junto a otras con alguna característica extra:

- smallv: aplicación básica, en ella se centra el documento
- smallvcal: añade características de calibrado
- smallvmat: añade compatibilidad con MatLab
- smallv640: utiliza ventanas de 640x480 en la interface gráfica

Este es el aspecto que tiene al arrancar la aplicación básica:



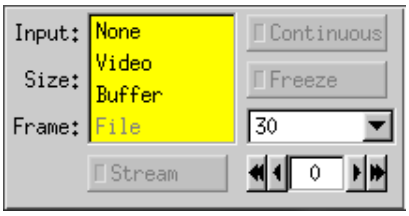
Si la cámara ha sido detectada deberá aparecer al pulsar el menú *Device* tal que así.



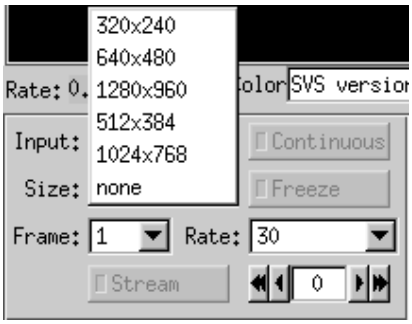
Iniciar la transmisión/recepción

Para que la cámara empiece a transmitir hay que seguir los siguientes pasos:

1. Desplegar el menú de opciones de *Input* y seleccionar *Vídeo*.



2. Desplegar el menú de opciones de *Size* y seleccionar la resolución que interese. Se ha comprobado que la aplicación solo permite seleccionar 320x240 y 640x480 a 30Hz.



3. Si la cámara está detectada y todo ha salido bien, la aplicación permitirá pulsar el botón *Continuous*. Con esto el led de la cámara empezará a parpadear y saldrá la imagen en las ventanas superiores, que hasta ahora estaban en negro.

4. Llegado a este punto, puede que no se vea nada o que se vea mal. Cada objetivo de la cámara se puede regular, tanto en iluminación como en enfoque, por lo que hay que configurarlos de forma individual comprobando el resultado en pantalla.

5. Para cambiar la resolución, y que sea efectivo, se tiene que parar la captura volviendo a pulsar el botón *Continuous*. Tras cambiarla ya se puede volver al paso 3.

Opciones de visualización

Cuando la cámara esté transmitiendo se mostrará una imagen en cada ventana, correspondiendo a la imagen del objetivo izquierdo la de la ventana izquierda, y la del objetivo derecho la de la ventana derecha.

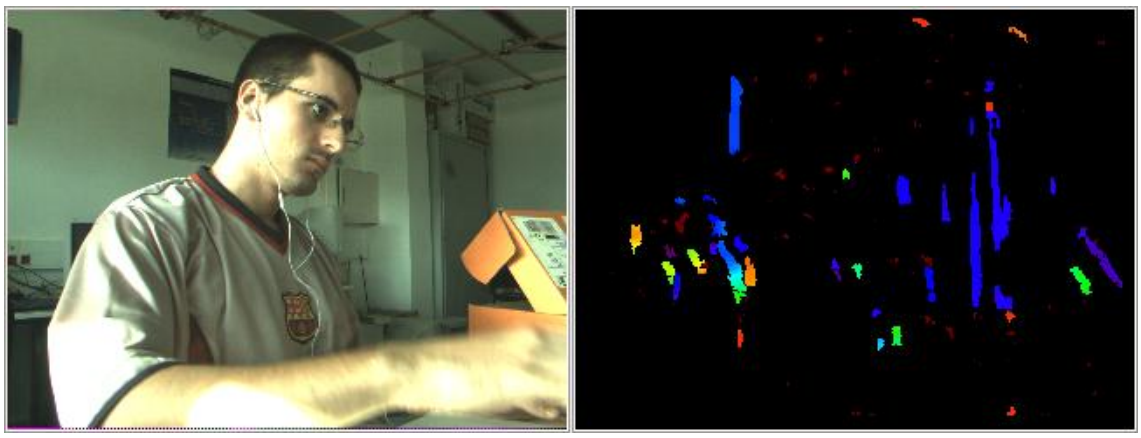
Existen algunas opciones interesantes que modifican lo que se muestra en las ventanas:

- Botón *Left*: desactivado bloquea la retransmisión de la imagen de la ventana izquierda.
- Botón Right: actúa como el botón Left pero en la ventana derecha.
- Botón *Color*: muestra la imagen que tiene encima en color o escala de grises.
- Slider *Gamma*: rectifica la gama de colores de las dos ventanas.
- Botón *Freeze*: congela las imágenes.
- Desplegable *Function*: *Stereo* activa la vista de profundidad en la ventana derecha.

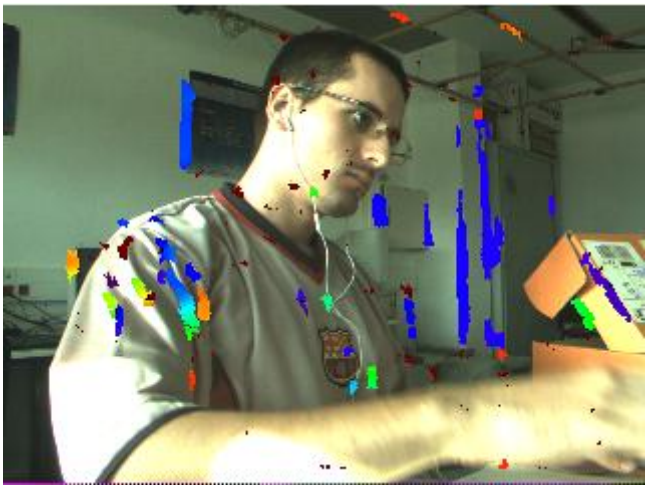
Detección de objetos

La aplicación permite un modo de visualización teniendo en cuenta la profundidad de los elementos captados por la cámara. Primero, para activarlo hay que seleccionar la opción *Stereo* del menú desplegable *Function*.

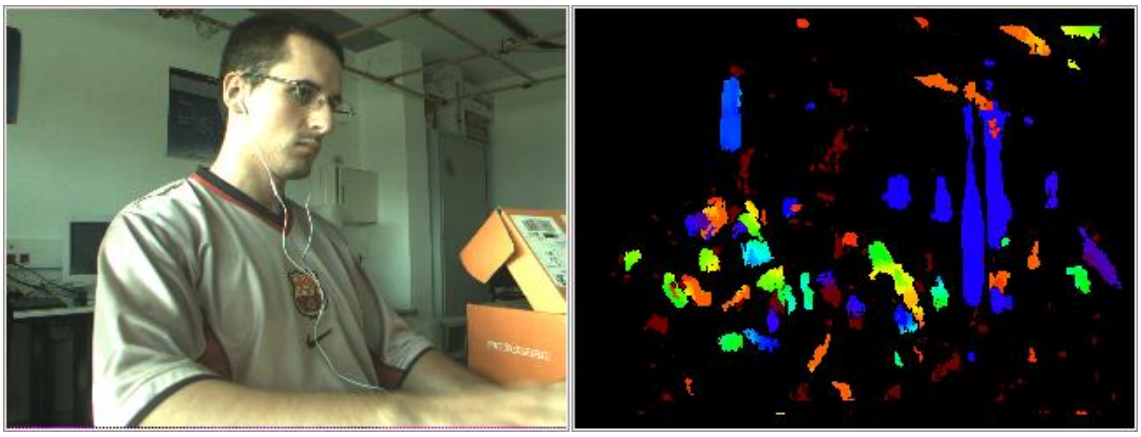
Una vez seleccionado, en la ventana de la derecha aparece una interpretación de la profundidad de los objetos de la escena, en una gama de colores, según sea su distancia con el objetivo.



Vista la imagen, queda patente que la detección deja bastante que desear, aunque superponiendo las imágenes se demuestra que al menos, lo que sale, son detalles significativos de la imagen.



Dentro de las opciones de la aplicación, hay una que amplía el rango de detección y muestra más detalles en pantalla. Se sitúa en la parte inferior derecha de la ventana y es un botón con el título *Multi Scale*. Al activarlo se consigue este efecto.

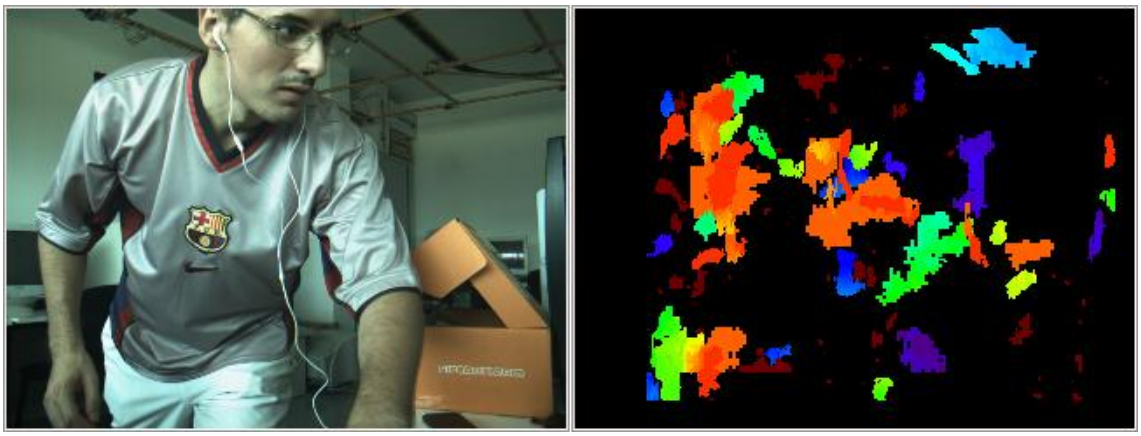


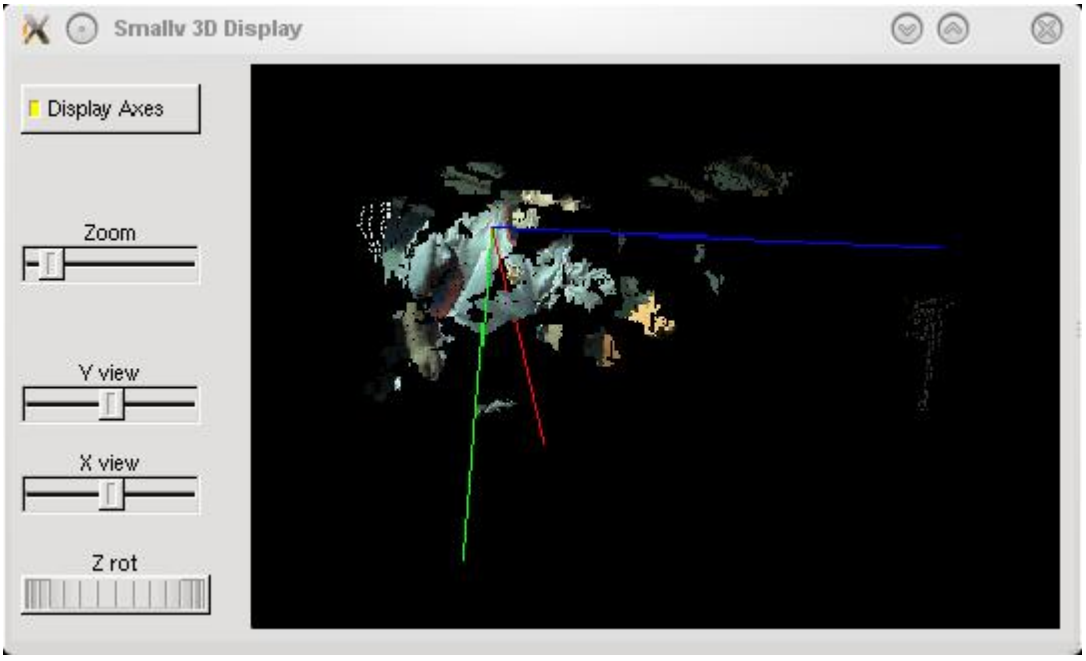
La mejora no es ideal, pero si amplia el tamaño de las muestras.

La cantidad de detalles que se muestran varían mucho según la posición, por lo que hay que ir probando.

Dentro de las opciones, hay una bastante espectacular. Realiza una reconstrucción, sobre un eje tridimensional, de los objetos detectados aplicando el color que les corresponde. De modo que queda un modelo 3D de la escena que se puede rotar, dentro de lo que permiten las capacidades de la aplicación al captar profundidades.

Para que se muestre la recreación de la escena en 3D, hay que pulsar el botón *3D Display*. No se va actualizando, por lo que se tiene que pulsar cada vez que se quiera refrescar el modelo 3D. La escena se muestra en una ventana nueva, donde hay varias opciones para rotar los ángulos y hacer zoom.



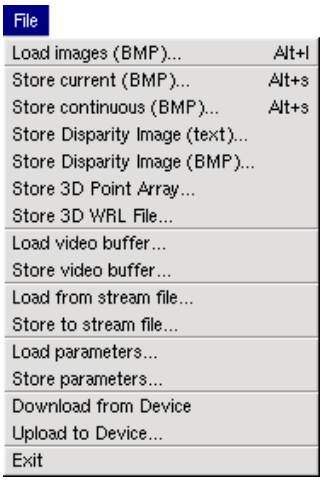


Este es un ejemplo de lo que se puede conseguir con esta opción. Aunque para conseguir una representación tan detallada se necesitan hacer bastantes intentos. Es interesante ver que si se cambia el ángulo poco a poco la imagen se va desfigurando, aunque deja cierto margen donde se puede reconocer la escena.

Guardar capturas

La aplicación tiene varias opciones para guardar capturas dentro del menú *File*, aparte de otras para cargar capturas guardadas anteriormente.

STORE CURRENT (BMP): guarda la imagen de cada ventana generando 5 archivos.



- Imagen-C.bmp es la imagen izquierda en color
- Imagen-L.bmp es la imagen izquierda en B&N
- Imagen-Q.bmp es la imagen derecha en color
- Imagen-R.bmp es la imagen derecha en B&N
- Imagen.ini guarda los parámetros de la captura

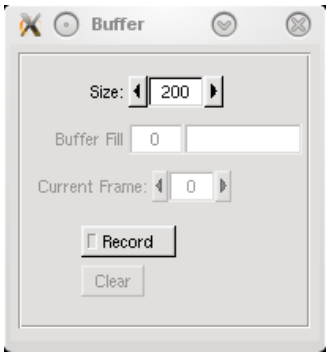
STORE DISPARITY IMAGE (TEXT): devuelve un documento de texto, con los valores numéricos de cada pixel de la imagen de detección de objetos. A la práctica, cada pixel donde no se detecta nada se marca con -1, y si se detecta algo, se marca con un número bastante más alto.

STORE DISPARITY IMAGE (BMP): guarda la imagen de detección de objetos en escala de grises.

Guardar secuencias

El mismo menú *File* comentado en el punto anterior tiene un par de opciones para guardar secuencias de imágenes.

STORE VÍDEO BUFFER: guarda la secuencia de imágenes capturada en el buffer como si fuesen imágenes individuales. De este modo, cada "imagen individual" supondrá 4 imágenes y un archivo, como los comentados en el punto *Store current (BMP)*. Como ejemplo, si el buffer fuese de 100 frames tendríamos:



- 100 imágenes izquierdas en color
- 100 imágenes izquierdas en escala de grises
- 100 imágenes derechas en color
- 100 imágenes derechas en escala de grises
- 100 parámetros, uno por captura

El menú del buffer se abre pulsando en *Vídeo Buffer*, a la derecha de *File*. En este, *Size* indica la cantidad de frames a guardar, *Record* es el botón para iniciar el llenado y *Clear* el botón para vaciarlo.
STORE TO STREAM FILE: genera un archivo con extensión .ini con los parámetros de la captura y otro .ssf donde se van guardando las capturas. El problema es que peta al querer abrirlo con la aplicación, así que no se ha podido comprobar, pero el archivo va creciendo como toca a medida que pasa el tiempo.

E.5 Cosas a tener en cuenta

Aplicación

RESOLUCIÓN DE CAPTURA:

La aplicación permite cambiar la resolución de captura pulsando solo *Freeze*, pero el cambio no se hará efectivo en realidad. Las capturas que se guarden, tras cambiarla de este modo, tendrán la resolución que tenía antes de realizar el cambio.

La cámara indica en sus características que permite resolución máxima de 1280H x 960V píxeles. A la práctica, a 30Hz la aplicación solo acepta las resoluciones 640x480 y 320x240, para conseguir 1280x960 hay que bajar a 7.5Hz en el menú desplegable *Rate*.

RATIO DE CAPTURA:

La aplicación tiene un indicador debajo de la ventana izquierda. El valor va oscilando alrededor de 25fps, aunque si se utiliza la función *Stereo* cae. Se supone que es porque no da más de si la aplicación, ya que está interpretando cada frame que le llega.

3D DISPLAY

La ventana se puede hacer tan grande como se quiera, y se puede rotar la imagen con el ratón.

A la hora de conseguir una “buena” imagen tridimensional, es necesario buscar la posición de modo que aparezcan más “manchas de colores” en la ventana de detección de objetos. Esto es porque son esos objetos los que se utilizan luego para crear la figura 3D. Por esto también es recomendable hacerlo siempre con la opción *Multi Scale* activada.

Suele costar conseguir la posición de los ejes que permita entender la imagen, pero una buena opción es colocar:

- El vector rojo lo más perpendicular a la pantalla posible (que se vea lo mínimo de este vector). Que el lado que quede más cerca sea en el que nacen los otros vectores, no al revés (que estén al fondo).
- El vector azul paralelo al borde inferior de la pantalla.
- El vector verde 90º a la derecha de este.

Cámara

RECONOCIMIENTO DEL SISTEMA:

En caso de que la aplicación no reconozca la cámara, pero el sistema sí, hay que asegurarse de que el usuario tenga permisos para utilizar el puerto IEEE 1394. Una prueba posible es intentarlo como root, así debe funcionar siempre.

ALIMENTACIÓN:

No tiene alimentación extra, por lo que se alimenta directamente del puerto IEEE 1394. Por este motivo es recomendable usar un cable de 6 pines en ambos extremos, no los típicos de portátil con conexión de 4 pines en un extremo. No se ha probado con los de 4 pines, pero no debería bastarle ya que no alimentan apenas.

ANEXO F: SIP

Session Initiation Protocol (SIP), o Protocolo de Inicio de Sesiones, es un protocolo desarrollado para ser el estándar para iniciar, modificar y finalizar sesiones multimedia [S1]. Se utiliza en una gran variedad de aplicaciones, como reproducción de telefonía IP, mensajería instantánea, vídeo, juegos, etc. Cualquier sistema que requiera de la negociación entre dos o más partes, se puede gestionar mediante SIP.

En este proyecto se ha intentado integrar, para gestionar los clientes reproductores con el repetidor, y no tener que utilizar el sistema de mensajes que al final se ha integrado [anexo D]. Los principales problemas han sido la imposibilidad de encontrar una librería sencilla. La gran mayoría están destinadas a telefonía IP, que es donde más está integrado el SIP, y otras han resultado ser demasiado complejas y difíciles de modificar para adaptar a nuestras necesidades. A continuación hay una descripción de varias librerías estudiadas con los problemas que ofrecieron, que puede servir de guía de cara a futuros proyectos fin de carrera.

GNU oSIP (Open SIP) [S2]:

Esta librería es de GNU y está implementada en C. Permite realizar desde clientes hasta servidores SIP. Se pueden encontrar ejemplos de cómo se utiliza pero están demasiado dirigidos al uso de VoIP. Realmente es muy amplia y conociéndola al detalle se podría hacer cualquier cosa con ella, pero es demasiado compleja para realizar una adaptación a corto plazo.

eXosip [S3]:

Esta librería está basada en oSIP. Facilita la implementación de clientes pero está demasiado dirigida a VoIP aunque, según dicen, con ella se puede hacer cualquier otro tipo de cliente. Los ejemplos que aparecen vuelven a ser centrados en VoIP y no ayudan a la hora de querer modificarlo para nuestros requisitos.

Live555 Streaming Media [S4]:

Live555 son un conjunto de librerías escritas en C++ preparadas con todo tipo de utilidades, siguiendo los estándares (RTP/RTCP, RTSP, SIP), para streaming multimedia. Ofrece la posibilidad de crear aplicaciones para streaming de audio y vídeo, servidores de contenidos multimedia y clientes y servidores SIP. Es tal su oferta que el módulo SIP no es ni mucho menos su punto principal, y encontrar un ejemplo para ver cómo funciona no ha sido factible, aunque teóricamente no tiene que ser demasiado difícil. Se ha intentado realizar pero no se ha logrado, habría que estudiar mejor la librería.

Javax SIP [S5]:

Java ofrece una librería para procesar mensajes según el protocolo SIP, pero usarla sin ningún ejemplo es inviable. El hecho de ser en java y que todo el proyecto esté en C ya condiciona su utilidad así que se prefirió buscar ampliaciones funcionales que la usasen.

Cliente SIP de i2Cat [S6]:

Esta es una aplicación de i2Cat en java, que utiliza las librerías javax sip comentadas en el punto anterior. Aunque no esté terminada, está pensada para conectarse a un servidor, darse de alta, gestionar ciertas configuraciones de estado y realizar streaming de vídeo. El principal problema es que, al ser una aplicación con tantas funciones, es muy complejo extraer la parte que nos interesa para el proyecto, aparte de no saber seguro si funcionará bien.